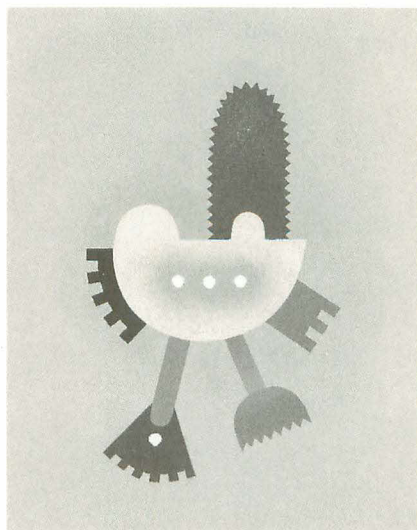
SX-WINDOW ver.3.1
開発キット

著●吉沢正敏・牛島健雄・西田文彦・小浜 純

Cover Illust.=Shuuhei EGUCHI
Cover Design=Masaki KATSUMATA

SX-WINDOW ver.3.1
開発キット

著●吉沢正敏・牛島健雄・西田文彦・小浜 純

はじめに

SX–WINDOW の本格的な解析資料として『SX–WINDOW プログラミング』、『追補版 SX–WINDOW プログラミング』が刊行されたのが 1991 年。あれから早くも 4 年の歳月が過ぎようとしています。当時はまだ実用面で不満な部分もあった SX–WINDOW も、現在では機能的にも充実した ver.3.1 がリリースされ、シャープからの公式な開発資料として「SX–WINDOW 開発キット Workroom SX–68K」が発売されるなど、SX–WINDOW を取り巻く状況は、4 年前と比べて飛躍的に向上したといえるでしょう。

しかしながら、全国の SXer (SX–WINDOW でプログラミングを目指す人々の呼称) の期待を一身に背負って発売された Workroom の内容は、3 年前の ver.2.0 相当の情報を公開するに留まり、SX–WINDOW プログラミングの開発環境は時代に取り残された形となっています。

そのような現状を打破し、追補版からの空白の時間を埋め合わせ、SX–WINDOW ver.3.1 に対応した最新の開発環境を提供するとともに、『SX–WINDOW プログラミング』、『追補版 SX–WINDOW プログラミング』、そして Workroom SX–68K と分散してしまった SX コール・リファレンスを 1 つにまとめ、SX コールの集大成としても機能するようにと本書は企画されました。

本書は、主に C 言語で SX–WINDOW アプリケーションを開発する人を対象としたプログラミング入門、提供ライブラリなどの開発環境の使い方の解説からなる第 1 部と、ver.3.1 まで対応した SX コールの総集編である第 2 部から構成されます。

大まかな内容は次のようになっています。

第 1 部　SX–WINDOW 開発入門

　第 1 章　SX–WINDOW プログラミングの基礎

　　これから SX–WINDOW でのプログラミングを始めようという人を対象に、その基礎をおさらいし、これまでの SX–WINDOW プログラミング環境の流れについて解説します。

第2章　インストール

添付フロッピーディスクから本書の筆者たちが推奨する開発環境をインストールする方法と、添付CD-ROMからSX-WINDOW用に開発されたフリーソフトをインストールする方法について説明します。

第3章　SX-WINDOW ver.3.1 開発キット

Workroomを基礎としてver.3.1対応を行ったSX31KITの概説と、SX31KITによるプログラミング、callno headerの使い方についてサンプルをまじえて解説します。その他、ver.3.0になって新設されたマネージャの機能と簡単な利用方法を紹介します。

第4章　LIBSXC

X680x0のgccでSX-WINDOWアプリケーションの開発を行うにあたって必要不可欠なライブラリであるLIBSXCについて詳説します。

APPENDIX

SX31KITの構造体の詳細やLIBSXCの内部構造などについて中級者以上向けに解説してあります。

さらに、添付CD-ROMに収録させていただいたフリーソフトの一覧を簡単な紹介とともに掲載してあります。

第2部　SXコール・リファレンス

SX-WINDOW ver.3.1に対応した全コール・リファレンスを記載しました。

添付フロッピーディスク

本書で提供する基本的な開発環境(LIBSXC, callno header, SX31KIT)、ならびに株式会社計測技研のご厚意によりCD-ROMドライバ(機能限定版)を収録しました。

インストールおよび収録内容についての詳細は「2.1　添付FDからのインストール」(p.34)をご覧ください。

添付 CD–ROM

大手 BBS である NIFTY–Serve と、SX–WINDOW を中心とした草の根 BBS である Network–SX NG から集めうるかぎりの SX–WINDOW フリーソフトと gcc や mule などの開発ツール一式を収録しました。いずれも秀作ばかりを贅沢に揃えましたので、CD–ROM ドライブを用意してぜひお楽しみください。また、一部のフリーソフトにつきましては作者のご厚意によって、ソースファイルも添付させていただきました。SX–WINDOW でのアプリケーションの作成にお役立てください。

なお、収録されているフリーソフトの一覧については APPENDIX C 章「SX–WINDOW フリーソフト一覧」 (p.171) を、インストールについての詳細は「2.2　添付 CD–ROM からのインストール」 (p.51) をご覧ください。

本書が SX–WINDOW 上にあなたの窓を開く助けとなることを祈って。

1995 年 6 月吉日　　ひそやかに梅雨のふる朝に

著者代表　牛島健雄

CONTENTS

第 1 部

SX-WINDOW ver.3.1 開発環境

第1章

SX-WINDOW プログラミングの基礎

SX-WINDOW は X680x0 シリーズで動作する疑似マルチタスク・ウィンドウシステムです。本章では、これから SX-WINDOW のアプリケーションの開発を始めようという人のために SX-WINDOW システムについて簡単な解説を行い、本書で推奨するアプリケーション開発環境について紹介します。

SX-WINDOW 概説

SX-WINDOW は、X680x0 シリーズの OS (オペレーティングシステム、基本ソフトウェア) である Human68k 上に構築されたマルチウィンドウシステムの総称です。イベント駆動型と呼ばれる方式による疑似マルチタスク方式を使って複数のアプリケーションを同時に実行することが可能です。

これらは、ウィンドウ環境を実現するための基本サブルーチン群を提供するソフトウェア "FSX.X" と、疑似マルチタスクによる動作環境を提供するためのソフトウェア "SXWIN.X" の 2 つから構成され、主に前者が "SX システム"、後者が "SX シェル" と呼ばれています。

本節では、SX-WINDOW の動作の仕組みと SX システムが提供するサービスについて説明を加えたあとで、SX-WINDOW 上で動作するアプリケーションのすべきこと、してはならないことについて述べます。

疑似マルチタスク処理

SX-WINDOW 上で実行される各々のアプリケーションはそれぞれが「タスク」と呼ばれ、SX シェルにおける処理単位として他の処理とは独立かつ並列に実行されます。

ここで SX-WINDOW で採用されているイベント駆動型と呼ばれる疑似マルチタスク処理はどのようなものなのか見てみましょう。

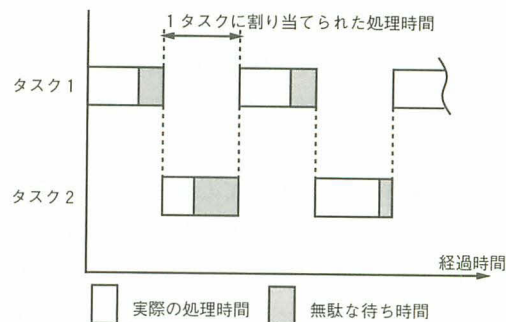
マルチタスク処理の基本は、原理的に 1 命令しか実行できない CPU において、あるタイミングで処理を切り替えることにより、個々の命令レベルではなく、各タスクの処理レベルから見た場合に複数の処理を同時に実行しているかのように見せることです。

一般にマルチタスクの実現方式は、そのタイミングのとり方によって以下の 2 種類に大別されます。

○ タイムスライシング型マルチタスク

○ イベント駆動型マルチタスク

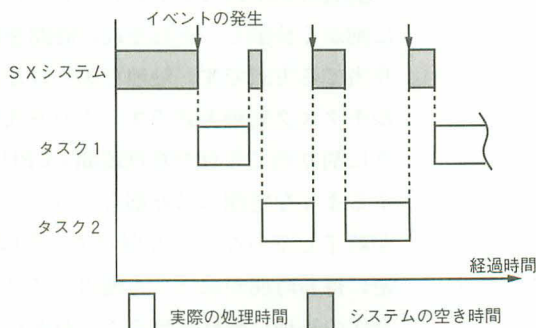
前者のタイムスライシング型マルチタスクは、CPU の動作を短い時間に細かく分割し、それぞれの時間を複数のアプリケーションの実行に割り当てる方式です。処理を切り替えるタイミングとして時間を用いたマルチタスク処理方式です。この方式の場合、原則としてそれぞれのタスクに割り当てられた処理時間 (CPU 時間) は一定なので、等間隔で実行するような処理には好都合なものの、持ち時間が過ぎれば現在のタスクが終了していなくても別のタスクに処理が切り替わってしまいますし、逆に持ち時間が余っても他のタスクに切り替わることがないなど、CPU 全体の処理に無駄が出ることがあります (Fig. 1-1)。そのため、処理速度の遅い CPU で実現されるマルチタスクシステムにはあまり利用されません。

Fig. 1-1 ● タイムスライシング型マルチタスクの動作

後者のイベント駆動型マルチタスクは、キーが押されたなどの、外部からのイベントに応じてタスクを切り替えるマルチタスク処理方式です。この方式では、それぞれのタスクは自分に関わるイベントの発生をつねに監視しており、それが発生した時点で必要な処理を行うことになるため、無駄な時間を費すようなタスク処理を未然に防ぐことができます。そのため、比較的処理速度の遅い CPU におけるマルチタスクシステムには最適の方式であるとされます。しかし、正確には複数のタスクが同時に実行されるわけではないため、「疑似マルチタスク」と呼ばれています (Fig. 1-2)。

SX-WINDOW ver.1.0 が発表された当時の X680x0 シリーズで採用されていた CPU は 10MHz の MC68000 という、それほど処理速度の速くない CPU であったことから、SX-WINDOW がイベント駆動型の疑似マルチタスク方式を採用したことは正しい選択といえました。

しかし、ここで注意しなければならない点が 1 つあります。

タイムスライシング型マルチタスク方式では、処理の切り替えはマル

Fig. 1-2 ● イベント駆動型マルチタスクの動作

チタスク動作を実現するシステムが自動的に行うのに対して、イベント駆動型マルチタスク方式では、タスクの切り替えタイミングは個々のタスクに依存しているため、イベントへの対応処理は比較的短時間ですまさなければならないということです。

つまり、実際の疑似マルチタスクの動作として、タスクからタスクへの切り替えは、タスクがイベントの処理を終了し、イベント待ちに入った時点で行われます。もし、あるタスクが受け取ったイベントの処理に時間がかかってしまうと、次に処理される予定のタスクは実行を待たされることとなり、システム全体としてはマルチタスクとはいえないような動作になってしまいます。SX-WINDOW では、システムの状態が変化するまで SX システムがずっと待機していることはなく、実際にはその間にアイドルイベントが発行されます。

このようなことから、イベント駆動型マルチタスクシステムで動作するアプリケーションでは各々のイベントに対応する処理を記述するだけでなく、マルチタスクを実現するための作法に則った記述を行わなければなりません。アプリケーションを記述するうえでの作法については「SX アプリケーションの基礎」(p.9)で詳しく述べます。

次に、SX システムが提供してくれるサービスについて解説します。

SXシステムが提供するサービス

SX-WINDOW は Human68k の上で動作していることはすでに述べたとおりですが、Human68k には COMMAND.X をはじめとするコマンド処理による対話型インターフェースのシェルしか用意されておらず、SX-WINDOW でのウィンドウやアイコンといったマウスオペレーションを基

本としたGUI (グラフィカル・ユーザ・インターフェース) のためのサービスは、すべてSXシステムによって提供されています。

これらのサービス群は「SXコール」と呼ばれ、メモリ管理、イベント処理、画面表示といったシステム内部で用いられるような基本的なものから、ウィンドウ描画、メニュー処理などの高度なものまで数多く用意されています。SXコールには、それぞれの役割をもとにして、メモリ管理を行うのがメモリマン、ウィンドウの描画はウィンドウマンというように、「～マン」という名称で呼ばれる、全部で21種類のマネージャが存在します。これらマネージャの一覧をTable 1-1に示します。

Table 1-1 ● SXシステムのマネージャ一覧

マネージャ名	処理の概要	備考
イベントマン	イベントの管理	
アニメーションマン	マウスポインタのアニメーションの管理	
コントロールマン	ボタンなどのコントロールの管理	
ダイアログマン	ダイアログボックスの作成や表示	
エクセプションマン	V–DISP割り込みの管理	
グラフマン	画面への描画	
キーマン	キーボードマンとイベントマンの仲介	
キーボードマン	キーデータの管理	
メモリマン	メモリの管理	
メニューマン	メニューの作成と表示	
マウスマン	マウスの管理	
リソースマン	リソースの管理	
タスクマン	全マネージャを統括し、タスク操作を管理	
テキストマン	文字列や文章の編集	
ウィンドウマン	ウィンドウの管理	
プリントマン	印刷機能の提供	ver.1.1より追加
サブウィンドウマン	サブウィンドウの管理	ver.1.1より追加
フォントマン	フォントの管理	ver.2.0より追加
カラーマン	カラー情報の管理	ver.3.0より追加
ビデオマン	映像再生機能の提供	ver.3.0より追加
セマフォマン	セマフォの管理	ver.3.0より追加

個々のマネージャの詳細については、『SX–WINDOWプログラミング』(以下、SX本と呼ぶ) ならびに『追補版SX–WINDOWプログラミング』(以下、追補版と呼ぶ。いずれもソフトバンク刊)、『Workroom SX-68Kプログラマーズマニュアル』(シャープ) などを参照していただくとして、ここでは説明を省略させていただきます (SX–WINDOW ver.3.0で新たに追加された、カラーマン、ビデオマンについては「3.2　SX–WINDOW ver.3.1環境でのプログラミング例」(p.92)などで取り上げていますので、そちらを参照してください)。

SX–WINDOWにおけるSXコールは、Human68kにおけるDOSコー

ルにあたるものですが、両者には決定的な違いがあります。

Human68k は、基本的にはアプリケーションが複数同時に動作することがないシングルタスク OS であるため、現在動作しているアプリケーションはメモリと画面の全部を好きなように使うことができる一入出力の系であるといえます。DOS コールは、そのようなアプリケーションから利用されることを前提としたサービスを提供しています。

それに対し、SX-WINDOW 上では複数のアプリケーションが同時に動作し、個々に画面出力があり、ファイル入出力があるといった、いわば多入出力の系となっています。そのため、SX コールでは GUI などのウィンドウシステム固有の処理を提供すると同時に、ウィンドウというアプリケーション固有の画面を提供し、文字列の描画はウィンドウ内へ、ファイルの処理は排他的に行うといった、DOS コールではほかに影響を与えてしまうような処理を保護する役割もしています。

また、SX-WINDOW は Macintosh などで提供されている基本システムとしてのウィンドウシステムとは異なり、SX シェルで提供される疑似マルチタスクシステムを Human68k 上の 1 つのアプリケーションとして実現しています。そのため、 IOCS コールや DOS コールにより疑似マルチタスクシステムを破壊するような行為は禁止されています。これは、SX-WINDOW の利点であるシステムの柔軟さを引き出していると同時に、システム保護の面からは最大の弱点であるともいえ、アプリケーションを作成するうえでつねに注意しなければならない点です。

以上のことから、SX システムで提供されるサービスには、疑似マルチタスクによるウィンドウシステム上で動作するアプリケーションに必要不可欠な要素が詰まっていることがおわかりいただけたかと思います。このことは、言い換えれば SX コールをうまく利用することによって、比較的簡単にアプリケーションの構築が可能であるということなのです。これら SX コールはアプリケーションの原石といえるのです。

それでは、SX シェル上で動作するアプリケーションが満たすべき点は何かということを次に考えてみましょう。

SXアプリケーションの基礎

Human68k アプリケーションとの違い

わたしたちが慣れ親しんでいる Human68k 上のプログラムと SX-WINDOW 上のプログラムとでは何が違うのでしょうか？

まず、前述したように Human68k は基本的にシングルタスクであり、SX-WINDOW は疑似ではあるがマルチタスク処理を可能としているという点が挙げられます。SX-WINDOW 上で動作するアプリケーションは、Human68k 上のそれとは違って画面を自由に使うことはできませんし、空きメモリを自由に使うといったこともできないかわりに、“他のタスクに影響を与えない範囲で”という制限付きの自由が与えられています。それはすなわち、アプリケーション固有のウィンドウを開くことであり、マルチタスクで動作する SX-WINDOW アプリケーションを作成するうえでの基本なのです(もちろん、ウィンドウを開かないアプリケーションが存在してもかまいません)。

次に、SX-WINDOW アプリケーションでの処理の流れが、Human68k 上で動作するものと違って、SX シェルからのイベント処理要求を待ち、受け取ったイベントの種類に対応した処理を行うといったイベント駆動型のプログラムでなければならないという点が挙げられます。

つまり、すべての SX-WINDOW アプリケーションにおいて、

(1) イベントを受け取るまで待機する。

(2) 受け取ったイベントに応じた処理を行う。

(3) 処理終了後は再びイベント待ち状態となる。

という基本的な流れと、マウスのボタンが押された、キーが押されたといった各種イベントへの対応という 2 種類の処理を記述しなければなりません。

たとえば、キーが押されたら、入力された文字を画面に表示する簡素なプログラムについて両者のプログラムスタイルを比較してみると、

Human68k

キーが押されるまでひたすら待ち、何か入力されたらその文字を表示する。

SX-WINDOW

ひたすらイベントを待ち続け、SX シェルからキーダウンイベントが発行されたら、その文字を表示する。

というようになります。

アプリケーション側から見た場合、キーが押されるまで待つという点では同じですが、システム側から見た場合には、Human68k ではアプリケーションの処理だけを行っているのに対し、SX-WINDOW ではアプリケーションがイベントを待ち続けている間に他の処理を行うことができる点に差があります。これがマルチタスクを実現する秘訣となっています。

マルチタスクプログラミングの実現

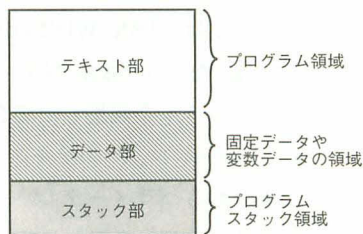
これらマルチタスクを実現するための機能は、マルチプログラミング方式という概念によっています。マルチプログラミング方式とは、メインメモリに複数個のプログラムを格納し、入出力待ちなどのタイミングでプログラムを切り替えることによって、コンピュータの利用効率を向上させる内部処理形態の1つです。

SX システムにおけるマルチプログラミングは、上で述べたイベント駆動型マルチタスクによる処理のほかに、メインメモリに複数個のプログラムを格納するための機構として、リエントラント (再入) 可能なタスク構造が導入されています。

ここで、リエントラントとは何かを説明する前に、SX-WINDOW におけるタスクの構造についてふれておきましょう。

SX-WINDOW における処理単位であるタスクは、一般に3つのメモリ領域として、テキスト部、データ部、スタック部を持っています (Fig. 1-3)。テキスト部は実際に実行されるプログラムのコードが格納されている領域であり、データ部はさらに初期化データ部と非初期化データ部 (bss) に分類されます (初期化データ部は固定データ、非初期化データ部は変数データを格納する領域であると考えてください)。

Fig. 1-3 ● タスクのメモリ構成概念図

テキスト部	プログラム領域
データ部	固定データや 変数データの領域
スタック部	プログラム スタック領域

実際に SX-WINDOW が動作している間は、このようなメモリ構成を持つタスクが SX シェルで管理されるメモリ領域内に複数個存在することになります。このとき、X680x0 のメインメモリが有限であることを考えれば、SX-WINDOW で利用できるタスクの数は個々のタスクがどれだけメモリを消費するかに依存していることは明らかです。つまり、タスクが消費するメモリをいかに節約するかが重要な要素となるわけです。

しかし、わたしたちが実際に SX-WINDOW で使用しているアプリケーションのなかには、エディタのようにプログラムそのものも大きく、大量にワークメモリを消費するようなものもあれば、"時計" のようにわずかしかメモリを消費しない小さなものまでさまざまなアプリケーションが存在しており、一口にメモリ消費量を抑えるといっても難しい問題です。

そこで、以下のような特別な条件を満たすアプリケーションを考えてみます。

1. 実行中にプログラムが変化しないこと。
2. 実行中にプログラムに必要な固定データが変化しないこと。
3. プログラムがデータに対して特定のアドレスに依存した形でアクセスしていないこと。

条件 1、2 は、いわゆる自己書き換えを行っているプログラムではないことの必要条件であり、条件 3 はレジスタ相対アドレッシング[1] と呼ばれる手法でのみプログラムが記述されていることを示しています。

この 3 条件を満たすプログラムが存在した場合、前述のタスクのメモリ領域のうちテキスト部は実行中に変更されることなく固定領域であるため、同じプログラムを複数回実行することがあってもテキスト部だけは共有できることになります (データ部、スタック部は新たに確保しなければなりません)。

このような条件に見合うタスク構造を持ったアプリケーションは、「リエントラント[2]可能」なタスク、もしくは OBJR 型タスク[3]と呼ばれ、タ

1) レジスタ相対アドレッシングについては XC アセンブラマニュアルなどを参照してください。本書の後半で詳しく述べますが、シャープ純正の C コンパイラ XC でコンパイルした場合は条件 3 が満たされず、フリーソフトのコンパイラである gcc の SX モードと呼ばれる特殊な方法でコンパイルしなければなりません。

2) テキスト部を共有したまま再度タスクとして入力できるという意味です。

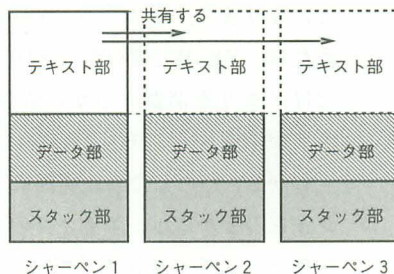
3) OBJR の R は "再び入る" という意味の英語 reentrant の頭文字です。

スクのメモリ消費量を抑えることができるよう実装されています。

たとえば、「SX-WINDOW ver.3.1 システムキット」に付属の“シャーペン.X”などが OBJR 型タスクの典型的な例です。“シャーペン.X”で複数のファイルを編集した場合、すべてのタスクは最初に起動されたタスク“シャーペン.X”のテキスト部を共有し、編集ファイルの内容や、編集にまつわるパラメータ(カーソルの位置やカレントフォントなど)をタスク別のデータ部として持っています(このときの各タスクのメモリ構成図を Fig. 1-4に示す)。

Fig. 1-4 ● OBJR 型タスクを複数起動したときのメモリ構成概念図

共有する

シャーペン1	シャーペン2	シャーペン3
テキスト部	テキスト部	テキスト部
データ部	データ部	データ部
スタック部	スタック部	スタック部

このほかにも、SX-WINDOW で実行可能なタスク構造として、以下の 3 種類が用意されています。

○ OBJC 型タスク

OBJC 型と呼ばれるタスクは、OBJR 型タスクにおいて先の 3 条件のうちいずれかが満たされていないテキスト部を有するアプリケーションとして位置付けられています。この型のタスクは、複数起動された場合にもテキスト部が共有されず、OBJR 型に比べてメモリの消費量は大きくなります。

○ OBJO 型タスク

OBJO 型と呼ばれるタスクは、OBJC 型に比べてさらに条件がきつくなっており、SX-WINDOW 起動中には 1 つしか動作できないという条件が付加されたアプリケーションです。例としては“GRW.X”のように、システムで 1 つしかない資源であるグラフィック画面を占有してしまうため、同じアプリケーションが複数起動しては困るような場合に用いられるタスク構造です。

○ OBJX 型タスク

OBJX 型と呼ばれるタスクは非常に特殊なタスクで、アプリケーション実行中に他のタスクが動作していては困るような場合に用いられます。現在、OBJX 型タスクとしては、唯一 “HD フォーマット.X” が存在します。

これらタスク構造に関するさらに詳しい説明は、『Workroom SX-68K プログラマーズマニュアル』(シャープ)、もしくは SX 本、追補版を参照してください。

● SX アプリケーションとイベント処理

SX 本、追補版でも述べたように、通常、SX シェル上で動作するアプリケーションが対応しなければならないイベントとして、以下の 11 種類のイベントがあります。

Table 1-2 ● 対応すべきイベントの種類

イベントの種類	イベントコード
アイドルイベント	E_IDLE
マウスレフトダウンイベント	E_MSLDOWN
マウスレフトアップイベント	E_MSLUP
マウスライトダウンイベント	E_MSRDOWN
マウスライトアップイベント	E_MSRUP
キーダウンイベント	E_KEYDOWN
キーアップイベント	E_KEYUP
アップデートイベント	E_UPDATE
アクティベートイベント	E_ACTIVATE
システムイベント 1	E_SYSTEM1
システムイベント 2	E_SYSTEM2

これらのうち、システムイベント 1 とシステムイベント 2 は同時に処理してもかまわないので、全部で 10 種類のイベントに対する処理だけでアプリケーションの記述が可能です。

以下、イベントと対応する処理内容について簡単に説明します。

○アイドルイベント

アイドルイベントは、SX シェルから定期的に発行されるイベントです。時間の表示やアニメーション処理といった、ユーザからの操作の有無にかかわらず動作し続ける処理を行うのに利用されます。

○マウスダウンイベント

マウスダウンイベントには、マウスの左ボタンが押されたときに発行されるマウスレフトダウンイベントと、右ボタンに対応するマウスライトダウンイベントがあります。

これらは、マウスのボタンが押されたことにより処理が発生する場合に利用されるイベントです。このイベントには、マウスのボタンが押されたという情報のほかに、マウスの位置、押されたウィンドウの情報が付加されており、アプリケーションは「自分のウィンドウで押されたか」「ウィンドウのどこの部分で押されたか」などについて知ることが可能です。

○マウスアップイベント

SX-WINDOW ver.3.1 の SX シェルでは、このイベントは発生しません。

○キーダウンイベント

キーが押されたときに発生するイベントで、押されたキーの内容が付加されています。キーボードによるショートカット・コマンドの処理も、このイベント対応処理で行います。

○キーアップイベント

SX-WINDOW ver.3.1 の SX シェルでは、このイベントは発生しません。

○アップデートイベント

他のウィンドウの下に位置していたウィンドウが上に移動した場合など、ウィンドウ内部を書き直す必要があるときに発生するイベントで、書き直すべきウィンドウの情報が付加されています。

SX-WINDOW では、ウィンドウ内部に描かれている情報についてSX シェルはなんら関知せず、ウィンドウを開いたアプリケーションが責任を持って描画を行わなければなりません。

○アクティベートイベント

あるウィンドウがアクティブになった、つまり、画面上で一番上のウィンドウになったときに発生するイベントで、アクティブになったウィンドウの情報が付加されています。

アクティベートイベントが発生すると、続いてアップデートイベントが発生しますので、このイベントでウィンドウ内部の再描画を行う必要はありません。あくまでも、一番上になったことを示すイベ

ントであることに注意してください。

○システムイベント

タスクマネージャや他のタスクが特別な通知を行う際に発生するイベントです。SX-WINDOW の終了やファイルの操作など、タスクの動作に影響する処理の前などに発生します。このイベントには、「システムイベントコード」と呼ばれる通知内容の詳細を示すコードのほかに各種の情報が付加されます。複数のタスク間で情報を交換する、タスク間通信などもシステムイベントとして通知されます。

システムイベント中で、すべてのアプリケーションが最低限対応しなければならないシステムイベントコードとして以下の 4 種類が挙げられます。

- NOTICEENDTSK
  タスクの終了予告イベントです。タスクが終了しては困る場合には、このイベントを取り除く処理を記述してください。
- ENDTSK
  タスクの終了イベントです。タスクを終了させなければなりません。
- CLOSEALL
  通常は、ENDTSK イベントと同じ意味を持っています。
- WINDOWSELECT
  ウィンドウがセレクトされたことを伝えるイベントです。ウィンドウを持っているアプリケーションでは、ウィンドウのセレクト処理を行わなければなりません。

実際にアプリケーションを作成する場合には、イベント待ちを基本とするイベントループ処理を骨格として、各種イベントへの対応処理を必要に応じて追加していくことで、見通しのよい効率的な開発が行えるでしょう。

まとめ

以上をまとめると、SX-WINDOW 上で動作するアプリケーションに最低限必要な項目として以下の点が挙げられます。

(1) アプリケーションの入出力は自分のウィンドウを開き、その中でのみ行うのが原則である。

(2) イベントループを中心とし、各種イベントに呼応する形式の処理体系で記述する。

(3) 受け取ったイベントへの対応処理は、できるかぎり短い時間で終了しなければならない。

(4) DOS コール、IOCS コールは原則として使用してはならない。

これらを遵守することは、SX-WINDOW 上で動作するアプリケーションとしての最低限のマナーであり、マルチタスクプログラミングへの第一歩なのです。

SXアプリケーション開発環境

SX-WINDOW ver.1.0 が発表されてからすでに 5 年の歳月が経過しており、その間に製品、フリーソフトを問わず多くの SX-WINDOW アプリケーションが登場しました。それらアプリケーションの開発は数多くの人たちによって支えられてきました。

「1.1　SX-WINDOW 概説」(p.4) では、SX-WINDOW のアプリケーションを作る場合のプログラム上の決まりについて述べましたが、本節では、SX-WINDOW のバージョンアップの歴史を振り返りながら、アプリケーションを構築するためのコンパイラ、ライブラリなどについて、紹介します。

本書で取り扱う“アプリケーション開発環境”とは、C コンパイラやアセンブラなどの開発言語ツールのほかに、C 言語のライブラリやインクルードファイルなど、アプリケーションを開発する際に必要なソフトウェアツール群全体を指すものとします。

SX-WINDOW の歴史

X680x0 シリーズ用疑似マルチタスク・ウィンドウシステム SX-WINDOW の歴史は、今から 5 年程前の 1990 年 4 月から始まります。

当時、X68000 シリーズには“ビジュアルシェル (VS.X)”[1]と呼ばれる SX-WINDOW の前身のような簡素なグラフィカル・ユーザインターフェースによるシェルが付属していましたが、実用性の面では“COMMAND.X”をビジュアル化した程度のものでした。X68000 SUPER の発売とともに登場した SX-WINDOW ver.1.0 により、本格的なウィンドウシステム時代の幕開けとなったのです。

その後、処理速度の向上を狙ってシステムクロックを 16MHz にした X68000 XVI の発売にともない、汎用性を持った小型ウィンドウであるサブウィンドウ機能と印刷機能を搭載し、処理の高速化を行った ver.1.10[2] が 1991 年 4 月 にリリースされました。

1) X68000 SUPER, EXPERT II, PRO II 以降のマシンには付属していません。

2) このほか、現在の“シャーペン.X”の前身であるテキストエディタ“エディタ.X”が新たに付属していました。

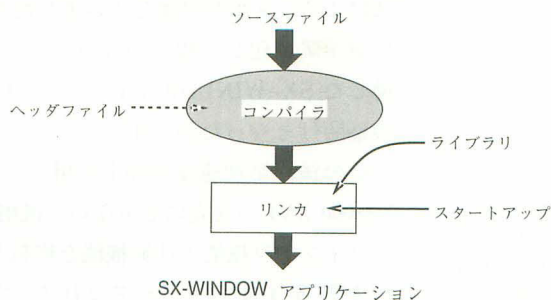
そして約1年後、X68000のシンボルであったツインタワーを廃止し、小さなボディを売り物にしたX68000 XVI Compact (コンパクト) の登場とともに、フォントマンの搭載によるアウトラインフォントへの対応、実画面モードの採用で画面スクロールが可能な広いデスクトップの提供、アイコンやメニューを編集するための各種コマンドの追加などによる、実用面での機能強化を目的としたver.2.00が1992年3月にリリースされました。

さらにその約1年後の1993年4月、X68000シリーズ最高峰の機種としてMPUに25MHzのMC68EC030を搭載したX68030が登場し、65,536色表示やビデオマンの搭載によるウィンドウ内部での動画表示をサポート、そしてマルチフォントによる高機能エディタ"シャーペン.X"が付属したver.3.0が発表され、大いに注目を集めました。

1994年5月にリリースされた最新版であるSX-WINDOW ver.3.1は、"シャーペン.X"のウィンドウ内でHuman68kのプログラムを動作させることが可能なコンソールモードの追加や、インラインかな漢字変換をサポートするなど、主に"シャーペン.X"の高性能化が行われています。

開発環境

SX-WINDOWのアプリケーションを開発するためには、SXコールを呼び出すためのヘッダとライブラリや、SX-WINDOWアプリケーションに独自の初期化処理を行うスタートアップルーチンが必要になります。また、OBJR型アプリケーションを作成することのできる開発ツールも必要でしょう。これらのヘッダとライブラリ、開発ツールを含めて「開発環境」と呼ぶことにします。

Fig. 1-5 ● SX-WINDOW アプリケーション作成の流れ

ここでは、SX–WINDOW 用の開発環境の歴史をたどったあと、なぜ、本書に添付されている開発環境が必要となったかを説明します。

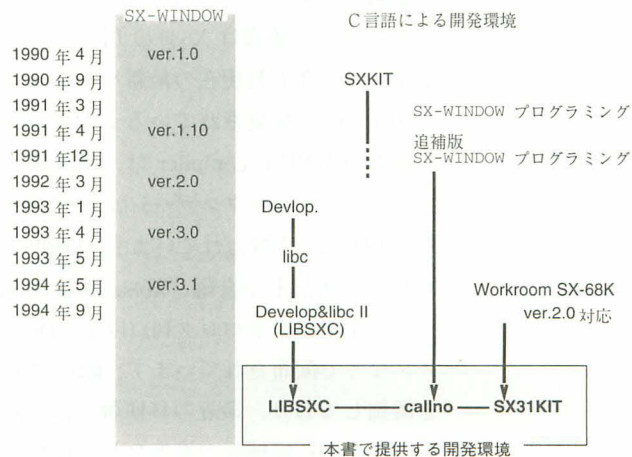
本書で提供する開発環境の位置づけ

SX–WINDOW のこれまでの開発環境には、メーカ (シャープ) 提供の開発キットである「SX–WINDOW 開発キット Workroom SX–68K」(以下、Workroom と呼ぶ) と、フリーソフト作者や書籍によって作成、公表されたさまざまな開発環境があります。

各開発環境の特徴については後述しますが、大きく分けると、フリーソフト作者たちが作り上げてきた開発環境とメーカ主導の開発環境とではソースプログラムレベルでの互換性がありませんでした。

また、Workroom の開発環境は SX–WINDOW の最新バージョンである ver.3.1 ではなく、ver.2.0 までしかサポートしていませんでした。本書で提供する開発環境は、Workroom の開発環境との互換性を維持しつつ、フリーソフトの開発環境の資産をも利用できる開発環境を SX–WINDOW ver.3.1 ベースで構築するために開発されました。

これまでの開発環境と本書で提供する開発環境との関係を簡単に図にまとめると、次のようになります。

Fig. 1-6 ● SX–WINDOW と開発環境の推移

このように、本書で提供する開発環境は開発環境の最新版であるとともに、それぞれの開発環境を統合して扱う開発環境になっています。

SX-WINDOW 開発環境の歴史

ここでは、これまでに公表されている主な開発環境にはどんなものがあり、その特徴は何かについて説明します。フリーソフトとして配布されているソースプログラムをコンパイルする際など、これらの歴史や関係等を知っておくと便利でしょう。また、本書で提供する開発環境がどんなものであるかを知るうえでも重要です。

ここでいう開発環境とは、C コンパイラ、ヘッダ(インクルードファイル)、ライブラリを指します。これら SX-WINDOW の開発環境について、どのようなものが今までに存在し利用されてきたのかについて以下順を追って概説し、最後に本書で提供する推奨開発環境について紹介します。

SX-WINDOW 開発環境(コンパイラ編)

SX-WINDOW アプリケーションの開発に利用されている C コンパイラは、以下の 2 種類に大きく分類されます。

○「X68000 C COMPILER PRO-68K ver.2.0 (もしくは ver.2.1)」

○ FSF のフリーソフトである「GNU C compiler」の移植版

前者は、シャープから発売されている X68000 用メーカ純正 C コンパイラであり、通常は X680x0 用 C コンパイラを略して XC と呼ばれています。1995 年 6 月現在の最新バージョンは ver.2.1 で、のちほど紹介する Workroom で推奨されている C コンパイラです[3)]。

3) X68030 に対応したライブラリが同梱されています。

後者の GNU C compiler は、Richard Stallman 氏らによる FSF(フリーソフトウェアファウンデーション)で開発された高性能 C コンパイラであり、通称 gcc と呼ばれています。X680x0 用として移植されている gcc は、パソコン通信上や書籍『X68k Programming Series (#0) X680x0 Develop. & libc II』(ソフトバンク刊)(以下、Develop. & libc II と呼ぶ)の添付ディスクなどで配布されています。gcc は最適化の性能において、前出の XC を凌駕しており、最近の移植版では SX-WINDOW アプリケーション開発のための特殊機能の追加なども行われています。今日では gcc のほうが一般的に使われているようです。

そのほか、Charlie 氏によって、C 言語、C++言語、Objective-C 言語をコンパイル可能な最新版 gcc ver.2.6.3 (通称 gcc2) の移植が行われており、こちらも gcc 同様、SX-WINDOW アプリケーション開発のための機能拡張が施されています (開発途上ではありますが、作者のご厚意により本書の添付 CD-ROM に収録させていただきました)。この gcc2 の登場によって、C++や Objective-C で SX-WINDOW アプリケーションを作成する人も見受けられるようになりました[4)]。

本書で提供する開発環境では、callno header と LIBSXC [5)]が gcc の利用を前提に開発されています。XC では SX31KIT しか利用できませんので、極力 gcc を利用したアプリケーション開発をお勧めします。

4) SX-WINDOW ver.1.0 の頃のアセンブラだけの開発環境に比べれば、なんという進歩なのでしょうか。

5) フリーのライブラリである LIBC を SX-WINDOW 用に拡張したライブラリ。詳しくは、第 4 章「LIBSXC」(p.127) を参照してください。

► gcc の SX-WINDOW 用拡張機能

gcc は無料で使用できるコンパイラですが、市販されているコンパイラよりも高性能なことから、ワークステーション用からパソコン用まで多くの人々に愛用されています。X680x0 に移植されたものは、本来の gcc が備えていた拡張機能に加えて、X680x0 の環境に特化した拡張機能が付加されています (gcc の拡張機能の特徴についてはコラムを参照してください)。

⊙コラム：gcc の拡張機能

gcc は、もともとワークステーションの世界で生まれ育ってきたコンパイラです。X680x0 にはパソコンとしてはかなり早い時期に移植が始まり、多くの人々の努力によって極めて高性能なコンパイラに成長しました。

gcc の特徴は、まず、ANSI 完全対応+αの文法をサポートしていることでしょう (XC は準拠をうたってはいますが、完全には対応していません)。しかしながら、過去の資産を活用できるよう、「伝統的な」文法 (K&R のような) で記述されたソースにも対応できるようになっています。これを切り替えるのが "*-ansi*", "*-traditional*" といったオプションです。

また、gcc は非常に効率のよい実行コードを生成することでも知られていますが、その最適化の方法もきめ細かく指定できます。"*-f*" で始まるレジスタ割り付けオプションを駆使することによって、コンパイル時の動作をきめ細かく指定できるのですが、どのような組み合わせが有効であるか、作成するプログラムの特性にあわせて見極める必要があります。

X680x0 版に特化した拡張機能として、本文では "*-SX*" オプションを紹介していますが、このほかにも多くの X680x0 用オプションが存在します。ここではとても紹介しきれませんので、ぜひ『Develop. & libc II』もお手元に置かれることをお勧めします。

ここでは、X680x0 版 gcc の SX-WINDOW 用拡張機能について紹介します。拡張機能としては、現在、以下のようなものが用意されています。

○ リエントラントなプログラム開発を支援する機能

gcc は "*-SX*" オプションをつけてコンパイルすると、すべての変数を A5 レジスタからのオフセットでアクセスするようなコードを出力します。

これによってコード部分が共有できる、つまり、リエントラントなコードを自動的に作成することが可能となります。この場合、変数は `.rdata` セクションに置かれます(セクションについてはコラムを参照してください)。

○ `remote` 宣言

前述のとおり、"*-SX*" オプションをつけた場合、通常の変数は A5 レジスタからのオフセットでアクセスされます。`remote` 宣言は、変数のアクセスの際のオフセットが 64K バイトを超える大きさとなる場合に、オフセットを 32 ビットで扱えるようにするための予約語です。

`remote` 宣言された変数は 32 ビットのオフセットによるアクセスが可能となりますが、MC68000 マイクロプロセッサのマシン語命令では 1 命令で対応することができないことから命令コードが複雑となり、結果的にその変数へのアクセスは遅くなってしまいます。この指定が行われた変数は `.rldata` セクションに置かれます。

○ `common` 宣言

"*-SX*" オプションによって、通常の変数はすべて A5 レジスタからのオフセットでアクセスされるようなコードが出力されますが、`common` 宣言を行った変数は、通常の `.data` あるいは `.bss` セクションに割り当てられ、直接アドレスを指定してアクセスするようなコードが生成されます。

○ `SXCALL` 宣言

SX システムで提供されるサービスである SX コールは、通常、ライブラリを呼び出す形式でコードが生成されるのに対して、この `SXCALL` 宣言を関数のプロトタイプに付加することで、SX コールが直接呼び出されるようなコードが生成されます。これによって、ライブラリを呼び出すオーバーヘッドが軽減され、より実行効率

⊙コラム：セクション

セクションとは、アセンブルの際に便宜的に使用されるコード/データの格納場所の区分のことです。

XC に付属する as.x や lk.x は、.text, .data, .bss といったセクションをサポートしています。これらはそれぞれ、コードを格納する領域、プログラム開始時に初期値を入れておく必要があるデータを格納する領域、そして初期値を入れておく必要がない変数を格納する領域を意味しています。アセンブラソースのなかで適宜セクションの指定を行うことで、それぞれのセクションに格納すべきコード、ないしはデータが適当にまとめられて配置されます。

Human68k 上で動作するアプリケーションの場合は、この程度のセクションの指定だけで用が足りたのですが、SX-WINDOW 上で効率よく動作するアプリケーションを作るためには、少し性格の異なったセクションが必要となります。

SX-WINDOW 上でメモリをもっとも上手に使って効率よく動作できるのは「OBJR 型モジュール」と呼ばれるタイプの実行形式です。OBJR 型モジュールはリエントラントなコードで作られていなければならず、変数などが固定されたアドレスに格納されることは許されません。それらのデータは「データエリア」に格納され、A5 レジスタをベースにしたオフセット指定でアクセスする必要があります。しかし、前述の.data, .bss といったセクションは、プログラムに与えられたメモリ中の固定されたアドレス領域を使用するためのセクションなので、ここでは適切ではありません。

このような背景から「あるレジスタをベースにしたオフセット指定でアクセスするデータを格納するためのセクション」が必要となります。

フリーソフトのアセンブラとリンカである、has.x, hlk.x では、上記のような相対アドレス指定でアクセスするために「相対セクション」を追加しました。

相対セクションにはいくつか種類がありますが、ここでは本文中に登場する .rdata, .rldata セクションについてのみふれることとします。その他のセクションの詳細については『X68k Programming Series (#1) X68000 Develop.』(ソフトバンク刊) を参照してください。

○.rdata セクション

.rdata セクションは、あるレジスタ (SX-WINDOW 用アプリケーションの場合、データエリアの先頭を指す A5 レジスタ) をベースに、64K バイトの範囲をカバーするためのセクションです。64K バイトという制限は、MC68000 のアドレスレジスタ相対アドレッシングで指定できるオフセットのサイズがワードであることに由来しています。MC68000 でも 1 命令でアクセスできることから高速なアクセスが可能です。

○.rldata セクション

.rldata セクションは、あるレジスタ (.rdata と同様に、A5) をベースに、2^{32}の範囲を指定でき、X68000 の全メモリ空間である 16M バイトの範囲をカバーするためのセクションです。オフセットをロングワードで指定するために 64K バイトという制限はありませんが、MC68000 の 1 命令ではアクセスできず、.rdata セクションに比べて若干アクセス速度が落ちます。

gcc は、これらの新しいセクションに対応したコードを自動的に作成しますので、gcc + has + hlk という組み合わせで開発を行うかぎり、開発者は特に意識することなく、リエントラントなコードを作成することが可能となります。

のよいプログラムを作成することが可能となります。詳細については、「callno header の仕様」(p.115) を参照してください。

► C 言語標準ライブラリ

さらに、C コンパイラで利用される C 言語用の標準ライブラリとヘッダファイル (インクルードファイル) についても言及しておきましょう。

C 言語用の標準ライブラリには、コンパイラに対応して以下の 2 種類があります。

○ XC ライブラリ

シャープ純正コンパイラ付属のライブラリです。すべてアセンブラで記述されているため、比較的高速なものとなっています。

○ LIBC

LIBC は、パソコン通信上や書籍『Develop. & libc II』の添付ディスクで配布されているライブラリで、gcc から利用されることを前提として作成されています。

かなり多くの関数が C 言語で記述されているため、ライブラリの内容に変更を加えることも容易です。また、ANSI C [6] の仕様を満たし、POSIX [7] ライブラリなどの仕様を可能なかぎり取り込んでいますので、UNIX で利用されているツールの移植の際に威力を発揮します。

LIBC は、PDS [8] であると宣言されており、バイナリだけでなくソースも公開されています。

いずれも SX-WINDOW 用のライブラリではありませんが、C 言語でプログラムを作成するうえで必要不可欠なものとなっています。

ここで注意しなければならないのは、上記ライブラリは基本的に Human68k のプログラムで利用するために作られたものであり、SX-WINDOW 上で動作するプログラムから利用されることはいっさい考えられていないということです。

SX-WINDOW では、Human68k とはメモリ管理やファイル管理の方法が異なるため、前出の XC や LIBC に含まれている多くの標準ライブラリ関数 [9] を利用することはできません。

6) 米国規格協会 (ANSI – American National Standards Institute) の委員会で承認、制定された C 言語の標準化案のこと。

7) 米国電気電子学会 (IEEE – the Institute of Electrical and Electronics Engineers, Inc.) が制定したオペレーティングシステムに関する標準化案のこと。

8) Public Domain Software の略。著作者が、著作権を行使しないことを宣言したソフトウェアのこと。

9) たとえば、メモリを確保する malloc 関数やファイルオープン動作を行う fopen 関数など。

そのため、これら Human68k 用 C 言語標準ライブラリを、SX-WINDOW アプリケーションからも利用可能とするようにさまざまな改良を行った標準ライブラリがパワーユーザの手によって作成されています。現在、確認されているものは以下の 2 種類で、それぞれ XC, gcc 用ライブラリを改良したものとなっています。

○ SXCLIB

SXCLIB(KUM 氏[10]移植) は、XC ライブラリをもとに SX-WINDOW 用の改良を行ったライブラリです。XC ライブラリと高い互換性を持っていますが、現在では古いものとなっており、ほとんど利用されていません。

10) 本書の筆者の 1 人である西田文彦氏。

○ LIBSXC

LIBSXC (Niggle 氏[11]、KUM 氏移植) は、Human68k 用標準 C ライブラリ LIBC をもとに SX-WINDOW 用の改良を行ったライブラリで、本書の添付フロッピーディスク、ならびに添付 CD-ROM に収録してあります。

LIBSXC は本書の筆者たちが推奨する開発環境の 1 つであり、LIBC と上位互換性を保っていることから、LIBC 同様、UNIX のソフトウェア資産を流用する場合などにも移植にともなう作業が軽減されます。

LIBSXC については、第 4 章「LIBSXC」 (p.127) で詳説します。

11) 多田知顕氏。草の根 BBS の Network-SX 初代シスオペ。アーカイブされたファイルに対して SX-WINDOW 上で普通のディレクトリと同様な操作を提供するフリーソフト ViSON などを発表しています。

SX-WINDOW 開発環境 (SX ライブラリ編)

ここでは、C 言語で SX-WINDOW アプリケーションを開発するにあたって、従来から利用されてきたライブラリを紹介します。

SX コールを呼び出すためのライブラリ環境は、標準となるシャープからの開発環境の提供が比較的遅かったため、多数存在します。これらのライブラリを大きく分けると、次の 3 つに分類することができます。

○ フリーソフト作者たちがまとめた SXKIT

○ 追補版の添付ライブラリ

○ Workroom

以下では、これらを個別に取り上げて、その内容と特徴について述べるとともに、本書で提供する開発環境との違いについて説明します。

▶フリーソフト作者たちがまとめたSXKIT

SXKITは、X680x0用のフリーソフトを精力的に作成してきた沖@沖氏[12)]ら[13)]が、FSX.XやSXWIN.Xを解析してまとめた開発キットです。

フリーソフトのため市販はされませんでしたが、パソコン通信上で最初に公開された開発環境であることと、C言語の開発環境であること、MacintoshのToolBox[14)]に近い構造体名や関数名を使用している点などが特徴といえます。

SXKITは有志による解析をもとにして構築されたライブラリですので、解析が終了していない部分[15)]についてはサポートされていません。構造体の名前や関数名がWorkroomや追補版とはまったく異なります[16)]ので、過去のソースを活用しようとする場合にはソースプログラムを大きく変更する必要があります。

SX-WINDOWが発表されたばかりで、資料などもいっさいなかった当時、SX-WINDOWでのプログラミングを志す人たちは、全員、このヘッダとライブラリを使用していたといっても過言ではありません。その後、より詳細な情報が本として出版されたので、一般には徐々に使用されなくなっていったようです。

なお、本書で提供する開発キットはSX31KITという名前ですが、SXKITとは特に関連性はありません。

SXKITから本書で提供する開発環境まで、その内容がどのように変わっていったか、その変遷をご覧いただくために、もっとも顕著に変わってきたEventRecord型の内容を紹介します。

SXKITでは、下記のように定義されています。

List 1-1 ● EventRecord 型の定義

```
1:  typedef struct {
2:          short what;
3:          int   message;
4:          int   when;
5:          Point where;
6:          short modifiers;
7:      } EventRecord;
```

これは基本的にToolBoxの定義と同一であり、SXKITがToolBoxを参考に作られていることを端的に示しています。

12) 沖 勝氏。フリーソフト作者としてさまざまなネットワーク上で活躍しています。SX-WINDOW上の代表作として、プログラミング中にバスエラーやアドレスエラーが発生しても回復可能にするSXerrorや、エディタのSXngの移植など、多数のフリーソフトを作成しています。

13) 沖 勝、田口毅、青木司、清水和久、田中和明の各氏や、その他多くの人たちが協力してSX-WINDOWの解析を行っています。

14) MacOSのSXシステムに相当する部分。

15) 使用頻度の低いコールや、シェル用の複雑なコールなどの解析は困難だったようです。

16) たとえば、Workroom/追補版でいうWMNew関数はNewWindowという関数名になっています。

►追補版の添付ライブラリ

追補版では、SX–WINDOW ver.1.1 までのアセンブラマクロ、C 言語のヘッダとライブラリが提供されています。

このライブラリとヘッダの SX コールの命名方式や構造体の型名が以後、比較的広く普及したようです[17)]。

17)『追補版 SX–WINDOW プログラミング』は、巷では“SX 本++”と呼ばれる場合があります。

SX コール名は ToolBox 風の名前とは異なり、コール名の先頭 2 文字がマネージャ名を意味するようになりました。コール名がマネージャごとに分類されたことから、プログラムを作る場合も読む場合も、その処理内容を理解しやすくなっています。

構造体の定義についても整理されています。イベントの情報を伝える event 型の定義を追補版のヘッダから抜粋すると、下記のようになります。

List 1-2 ● event 型の定義

```
1:  typedef struct event {
2:              short           eWhat;  /* イベントの種類       */
3:              long            eWhom;  /* イベントに関連した引数*/
4:              unsigned long   eWhen;  /* イベントの発生時（シス*/
5:                                      /* テム内部カウント）   */
6:              point_t         eWhere; /* マウスの座標（グローバ*/
7:                                      /* ル座標系）           */
8:              unsigned long   eHow;   /* 特殊キーの状態       */
9:      } event;
```

基本的に型名が (event のように) 小文字から始まっているのが特徴です。

追補版のヘッダとライブラリは長期にわたって使用されてきたことから、古くから SX–WINDOW でプログラミングをしているユーザの間では現在でも利用される場合があります。

► Workroom

シャープが提供した Workroom では、SX–WINDOW ver.2.0 までのアセンブラマクロ、C 言語のヘッダとライブラリが提供されています。

構造体の型名や SX コール名の標準となるはずのライブラリといえますが、関数名は追補版のライブラリと同じであるものの、構造体名や型名、定数名が追補版のものとは異なります。

このため、Workroom 発売以前から作られていた追補版のヘッダを用いたフリーソフトのソースファイルをコンパイルすることができません。追補版のヘッダから Workroom のヘッダへ対応するためのソースファイルの変更は大変な作業量となり、いまだに Workroom 形式への移行は進んでおらず、現在でも追補版形式のヘッダとライブラリ用のソースプログラムが見受けられます。

構造体名の違いを、Event 型の定義を例にとって示します (Workroom の SXDEF2.H から抜粋)。

List 1-3 ● Event 型の定義

```
 1:  struct Event {
 2:          short what;
 3:          union {
 4:              Window *win;
 5:              struct {
 6:                  short code;
 7:                  short ascii;
 8:              } key;
 9:              long data;
10:          } whom;
11:          unsigned long when;
12:          Point where;
13:          unsigned long how;
14:      };
```

まず、第 1 に Workroom では、型名 (たとえば Event など) は大文字から始まっているのが特徴です。

第 2 の特徴として、C 言語の文法により無理なく適合できるように工夫されていることが挙げられます。

イベントレコードは、what の値によってその他の要素の意味が変わってきます。そのため、SXKIT や追補版の定義では、キャストを駆使して C 言語の文法に適合させる必要があったのですが、Workroom では、もっと無理なく使用できるよう union を使った定義が行われています。

what の値が意味するイベントの種類によって、whom の意味するものはウィンドウレコードへのポインタ、キーコード/ ASCII コード、その他のデータといった 3 種類に読み換える必要があります。たとえば、アップデートイベントの場合、whom はウィンドウレコードへのポインタとして解釈し、キーダウンイベントの場合は上位ワードをキーコード、下位ワードを ASCII コードとして解釈する必要があります。

追補版の定義では、アップデートイベント時のウィンドウレコードや、キーダウンイベント時のキーコードの値を得る場合には、次のようなコーディングが必要でした。

○ ウィンドウレコード:

```
winPtr = ( window*)( event.eWhom);
```

○ キーコード:

```
keyCode = ( event.eWhom) >> 16;
```

一方、 Workroom では、それぞれ以下のように記述できます。

○ ウィンドウレコード:

```
winPtr = event.whom.win;
```

○ キーコード:

```
keyCode = event.whom.key.code;
```

C 言語のコードとしては明らかに Workroom のほうがスマートですが、どちらが使いやすいかはプログラム作成者の好みの問題でしょう。

型の名前も構造体名と同様で、先頭の 1 文字が大文字となっています。たとえば、ポインタやハンドルを表現する型は以下のような違いがあります[18]。

18) ちなみに、追補版の handle の定義は間違っています。

○ 追補版:

```
typedef void *pointer;
typedef void *handle;
```

○ Workroom:

```
typedef void *Pointer;
typedef void **Handle;
```

定数の名前は、Workroom になって定義されないようになったもの (SHORTCUT 等) や、新しく定義されたもの (CHRBTN_H 等)、名前が変わったもの (CI_CHRBTN 等) がそれぞれ多数あります。

本書で提供する開発環境

それでは、本書で提供する SX-WINDOW ver.3.1 対応ライブラリについて簡単に紹介しましょう。

SX31KIT

SX31KITは、SX–WINDOW ver.2.0までしか対応していないWorkroomを拡張することによって、ver.3.1に対応したアプリケーションを作成可能とする、ヘッダとライブラリのセットです。

内容的には、Workroomの上位互換となっており、Workroomを利用してコンパイルできるプログラムはそのままSX31KITでもコンパイルすることができます。ただし、追補版のヘッダやライブラリとは互換性がありませんのでご注意ください。

SX31KITについては、「3.1　SK31KITとは」(p.74)で詳説します。

callno header

callno headerは、Workroomおよび追補版のヘッダファイルと互換性のあるヘッダです。このヘッダでは、SX–WINDOW ver.3.1までのSXコールをライブラリを通して呼び出す形式ではなく、直接A-line trap命令をインラインアセンブラとして展開する形式で関数を宣言しています。

callno headerのヘッダファイルでは、型や構造体の定義が含まれていないため、追補版もしくはWorkroomのヘッダファイルが必要となります。このとき、どちらのヘッダファイルを併用するかによって、いずれかの互換開発環境として機能します。

callno headerについては、「3.3　callno header」(p.111)で詳説します。

LIBSXC

LIBSXCはすでに述べたように、Human68k用標準Cライブラリであ るLIBCに、SX–WINDOW上では使えない標準ライブラリ関数を同様な機能を提供するSXコールに置き換えるなどの処置を施した、C言語でのSX–WINDOWアプリケーションの開発をサポートするgcc用C言語標準ライブラリです。

LIBSXCについては、第4章「LIBSXC」(p.127)で詳説します。

開発環境の組み合わせ

以上、これまでの開発環境と本書で提供する開発環境の関係を解説してきました。

本書で紹介する開発環境は、「追補版ヘッダ&ライブラリ」、「Workroom」、「LIBC」といった X680x0 の代表的な開発環境を統合し、これまでのソフトウェア的資産や開発経験をすべて SX-WINDOW 用アプリケーションに注ぎ込めるようにすることを目標としています。

「XC ＋ Workroom ＋ SX31KIT」や、「gcc ＋ 追補版ヘッダ＋ callno header ＋ LIBSXC 」などのように、かなり自由に組み合わせて使用できるため、どのような使い方をすればよいのか混乱してしまうかもしれませんが、以下の説明を読んでいただくことで、あなたにとってのベストな環境がおのずと決まってくることと思います。

とりあえず、いくつかの推奨プランを示しておきますので参考にしてください。

○初心者コース

SX-WINDOW プログラミング初心者で、難しいことはともかく、“動く”SX-WINDOW アプリケーションを作りたいという方へのお勧めは、

「XC ＋ Workroom ＋ SX31KIT」

です。

Workroom を基本に据えることで、Workroom に用意されているサンプルプログラムをそのまま使って学習することができます。学習が進んだところで、SX-WINDOW ver.3.0 以上の機能を使いたくなったときにも対応できます。

環境構築の手間が最小限ですむのがポイントです。

○中級者コース

Human68k 用アプリケーションを gcc ＋ LIBC で作っているが、SX-WINDOW プログラミングの経験があまりない方へのお勧めは、

「gcc ＋ Workroom ＋ SX31KIT ＋ LIBSXC」

です。

gcc でのプログラミング経験をそのまま活かしながら、SX-WINDOW ver.3.1 まで対応したアプリケーションを記述できるのが利点です。

○上級者コース

Human68k 用アプリケーションを gcc + LIBC で作っていて、SX-WINDOW のアプリケーションでも妥協のないものを作りたい方へのお勧めは、

「gcc + Workroom + SX31KIT + callno header + LIBSXC」

です。

gcc でのプログラミング経験をそのまま活かしながら、SX-WINDOW ver.3.1 まで対応した高速なアプリケーションを作成することができます。ほとんど究極の形態ということができます。

○古参コース

今さら追補版環境を壊せない、という古参 SXer の方へのお勧めは、

「gcc + 追補版ヘッダ + callno header + LIBSXC」

です。思うように、SX-WINDOW ver.3.1 対応アプリケーションを作成してください。

まとめ

本書で提供する開発環境は、先人たちの積み上げてきた X680x0 と SX-WINDOW に対する理解と情熱の集大成ともいえます。とっつきにくかった生の解析情報はかみ砕かれ、消化された形でヘッダやライブラリのなかに吸収されました。

次章からは、実際にこれらをどのようにインストールし、使用するかについての解説を行います。インストールが完了し、それぞれの環境の概要が頭に入ったところでスタートラインとなるわけですが、ゴールへの道のりは決して遠くはないはずです。

さあ、窓を開けて SXer への一歩を踏み出しましょう。

第2章

本章では、添付 FD に収められた開発環境のインストール方法と CD-ROM ドライバの設定方法、および添付 CD-ROM に収録されている SX-WINDOW 対応フリーソフトのインストール方法について紹介します。

添付FDからのインストール

本書には、SX-WINDOW 用のアプリケーションを開発するための基本的なプログラムを収めたフロッピーディスク（FD）と、SX-WINDOW 用に作られたフリーソフトなどを収めた CD-ROM が添付されています。

本節では、そのうちの FD 内に収められたファイルをハードディスクにインストールする方法について説明します。

インストールの準備

FD から実際にインストールを始める前に、必要な環境設定やハードディスクの容量などを確認しておきましょう。説明をよく読んでインストールの準備をしてください。

何をインストールするか

添付 FD に収録されている「SX-WINDOW ver.3.1 開発キット」は、以下の 4 つのパッケージから構成されています。

(1) SX31KIT ライブラリパッケージ
SX-WINDOW ver.3.0/3.1 で追加されたマネージャに対応したインクルードファイルやライブラリファイルが含まれています。

(2) callno header インラインヘッダパッケージ
X680x0 gcc の SXCALL 機能を用いて、高速かつコンパクトな実行ファイルを生成するためのインクルードファイルが含まれています。SX-WINDOW ver.3.1 に対応しています。

(3) LIBSXC ライブラリパッケージ
Human68k 上のフリーの ANSI C 準拠ライブラリである LIBC を SX-WINDOW に対応させたもので、LIBC 上位互換のインクルードファイル、ライブラリファイルなどが含まれています。

(4) LIBSXC ソースコードパッケージ

LIBSXC のすべてのソースコードが含まれています。

なお、

SX31KIT を利用するためには Workroom が必要

です。

添付 FD には、「SX-WINDOW ver.3.1 開発キット」専用のインストーラを 2 つ用意しました。

1 つは INSTALL.BAT で、SX31KIT, callno header, LIBSXC の 3 つのパッケージをインストールするためのものです。これら 3 つのパッケージの間には若干の依存性があるため、個別にインストールするのは面倒であり、かつミスを引き起こす可能性があります。本書では、これら 3 つを「**推奨開発環境**」としていますので、以降、これら 3 つを同時にインストールするものとして説明を行います。

もう 1 つは INSTSRC.BAT で、LIBSXC ソースコードパッケージをインストールするためのものです。このソースコードパッケージは、前述の「LIBSXC ライブラリパッケージ」のソースコード一式を収めたものです。通常の使用ではソースコードは不要なため、インストーラを別に用意しました。

どこにインストールするか

次に、インストールに必要なディスク容量を確認します。

それぞれのパッケージをインストールすると、おおむね Table 2-1 に示したディスク容量を専有します。インストール先のディスクに、インストールしたいパッケージのインストールに必要なサイズ以上の空き容量があることを確認してください。また、callno header をインストールするには、パッケージの容量以外に、0.2M バイト程度の作業領域が必要です。

Table 2-1 ● インストールに必要なディスク容量

パッケージ	ディスク容量
SX31KIT ライブラリパッケージ	0.5M バイト
callno header インラインヘッダパッケージ	0.2M バイト (+0.2M バイト)
LIBSXC ライブラリパッケージ	1.0M バイト
LIBSXC ソースコードパッケージ	3.5M バイト

たとえば、本書の推奨開発環境のみをインストールする場合、SX31KIT, callno header, LIBSXC の 3 つですから、約 1.7M バイトの空き容量が必要となります。

これらのパッケージを用いて SX-WINDOW 開発環境を構築するためには、SX-WINDOW ver.3.1 システムのほかに gcc あるいは XC などのコンパイラや、アセンブラ、リンカといったツールが必要であり、フロッピーディスクベースでの構築は困難です。

したがって、本書ではハードディスク上での構築を行うことを前提とします。コンパイラなどを含めた開発環境一式をハードディスクにインストールする場合、約 20M バイト程度の空き容量があれば十分収まると思います。最近では、数百 M バイトのハードディスクが非常に安く手に入れられるようになってきていますので、この機会に大きめのハードディスクを用意されてはいかがでしょうか。

必要な環境の準備

SX31KIT, callno header, LIBSXC パッケージをインストールするには、X680x0 gcc あるいは XC が動作するだけの環境が整っていれば、特に新しい環境の設定は必要ありません。

ただし、LIBSXC ソースコードパッケージのインストールには、特別な環境の設定が必要です。ソースコードパッケージ内のファイルには、長いファイル名を持つものが含まれており、通常の環境でインストールするとファイル名が重複して正しくインストールできないことがあります。そのためソースコードパッケージをインストールする前に、ファイル名として 21 文字すべて認識させることができるプログラム "TwentyOne.X" [1] を常駐させる必要があります。ソースコードパッケージのインストールに必要な "TwentyOne.X" や環境設定についての詳しい説明は、「LIBSXC ソースコードパッケージのインストール」(p.46) であらためて行います。

1) "TwentyOne.X" は、Ext（川本琢二）氏作のフリーソフトです。本書の添付 FD 内に "TWONE.X" というファイル名にリネームされて収録されています。

ライブラリパッケージのインストール

インストールの準備が整ったら、いよいよインストール作業を行います。説明をよく読んで、間違いのないように注意してください。

インストーラの起動

まず、X680x0 本体の電源を入れ、Human68k のコマンドシェル[2]を起動します。なお、添付 FD にはシステムが入っていないので、直接 FD から起動できませんので注意してください。

2)“COMMAND.X”や“FISH.X”, “KSH.X” など。

もし SX−WINDOW が起動しているようであれば、一度終了させてコマンドラインに戻るか、“COMMAND.X” のアイコンをダブルクリックしてコマンドシェルを起動してください。SX−WINDOW からインストーラを実行することはできません。

次に、添付 FD を用意します。安全のため、必ずバックアップを取り、バックアップしたフロッピーディスク（以下、インストールディスクと呼ぶ）を使用するようにしてください。インストールディスクを用意したら、X680x0 本体のフロッピーディスクドライブの 0 番に挿入し、カレントドライブを、インストールディスクを挿入したドライブに変更してください。

たとえば、インストールディスクを挿入したドライブが C ドライブであれば、次のようにキーボードから入力します。画面の表示例中の「A:¥>」という表示はコマンドシェルのプロンプトです。

```
A:¥> C:
```

次にインストーラ[3]を起動します。次のように入力してください。

3)インストーラに使われている cat.x, cp.x, mv.x, rm.x のフルアーカイブは、添付 CD-ROM の ¥OMAKE ディレクトリに収録されています。

```
C:¥> INSTALL.BAT
```

インストーラが正常に起動すると、順次画面にメッセージが表示されます。必要に応じて、インストーラが設定項目についてたずねたり、確認を求めることがありますので適切に回答してください。なお、メッセージ中の “CTRL+C” とは、［CTRL］キーを押しながら［C］キーを押すことを意味します。

インストーラを起動した際に、「(失敗) 環境変数 temp が定義されていないようです。テンポラリディレクトリのフルパスを環境変数 temp に設定して下さい。」と表示された場合は、適当なドライブ (たとえば、A ドライブ) に適当な作業用ディレクトリ (¥TEMP) を作ります。そして、コマンドシェルから「set temp=A:¥TEMP」のように入力してから、インストーラを起動してください。

```
**********************************************************
*          SX-Window3.1 Develop Kit - Install Script         *
**********************************************************

今から付属フロッピーディスクのインストールを行ないます。
質問に正しく答えてください。

(質問) 付属フロッピーディスクのうち, どれをインストールしますか？
  A. SX31KIT ライブラリパッケージ (空き容量が 0.5MB 必要です)
  B. CALLNO インラインヘッダ      (空き容量が 0.2MB 必要です)
  C. LIBSXC ライブラリパッケージ  (空き容量が 1.0MB 必要です)

         A.   B.   C.
  <1>    ○   ○   ○
  <2>    ○   ○   --
  <3>    ○   --   ○
  <4>    ○   --   --
  <5>    --   ○   ○
  <6>    --   ○   --
  <7>    --   --   ○

  <9>  インストーラを終了する。

1 から 7, あるいは 9 のうち、どれかひとつを選択してください。

(回答) --->
```

まず、インストールするパッケージを選択してください。たとえば、すべてのパッケージをインストールするなら「**1**」、SX31KITのみなら「**4**」を入力します。

本書では、3つすべてがそろった環境を推奨開発環境としていますので、以降、「**1**」を選択した場合にそって説明します。「**1**」以外を選択した場合は、適宜、該当部分以外の説明を読み飛ばしてください。

SX31KITのインストール

まず、インストール先のディレクトリ名を入力します。インストール先のディレクトリには、なるべく何もない新しいディレクトリを指定してください。インストール先にすでにファイルがあるとインストーラが誤動作することがあります。

```
 それでは「SX31KIT ライブラリパッケージ」をインストールします。

(質問) どこにインストールしますか？
       インストール先のディレクトリ名をフルパスで入力して下さい。
       ただしパスの最後に ¥ や / を付けないで下さい。

       中断するには CTRL+C を押してください。

(回答) --->
```

たとえば、A ドライブの「¥usr¥lang」というディレクトリにインストールする場合は、次のように入力します。なお、ディレクトリ名の最後に“¥”や“/”などのパス区切り記号をつけないようにしてください。

```
(回答) ---> A:¥usr¥lang
```

もし、インストール先の空きディスク容量が足りない場合、インストーラは次のようなメッセージを出力して実行を中断します。空き容量を再確認したのち、最初からインストールをやり直してください。

```
(警告) 指定したディレクトリには「SX31KIT ライブラリパッケージ」を
       インストールするだけの空き容量がありません。

       もう一度確認して最初からインストールをやり直してください。

C:¥>
```

次に Workroom が正しくインストールされているかどうかを確認します。ここでは、インストール先が空であることを想定していますので、次のようなメッセージが表示されます。

```
Workroom のインストールを確認します。NG.

  (警告) Workroom が正しくインストールされていません。

  (質問) Workroom をインストールしますか？
         インストールするなら y、
         しないなら n を入力して下さい。

  (回答) --->
```

SX31KIT を利用するためには Workroom が必要ですので、まず、Workroom をインストールしてください。ここでは「y」を入力します。

```
(確認) Workroom をインストールします。
       ドライブ1に Workroom アプリケーションディスクを挿入した後、
       リターンキーを押して下さい。
```

フロッピーディスクドライブの 1 番に Workroom アプリケーションディスクを挿入したあと、リターンキーを押してください。Workroom のうち、必要なファイルが自動的にインストールされます。

```
Workroom を A:/usr/lang にインストールします。
```

Workroom のインストールが終了すると、SX31KIT のインストールを行います。

```
    それでは「SX31KIT ライブラリパッケージ」をインストールします。

 Mode      Size     Date     Time Name
d-rwx         0 95-02-04 11:49:24 A:/usr/lang/INCLUDE/
-arw-      9598 95-01-08 12:00:00 A:/usr/lang/INCLUDE/COLOR.H
-arw-      2604 95-01-08 12:00:00 A:/usr/lang/INCLUDE/MENU.H
-arw-     36457 95-01-26 01:37:42 A:/usr/lang/INCLUDE/SXGRAPH.H
                  :
                  :
                 中略
                  :
                  :
-arw-     26882 95-01-08 12:00:00 A:/usr/lang/INCLUDE/SXCALL.MAC
-arw-     42911 95-01-26 01:37:40 A:/usr/lang/INCLUDE/SXCALL.H
d-rwx         0 95-02-04 11:49:26 A:/usr/lang/LIB/
-arw-     25740 95-01-08 12:00:00 A:/usr/lang/LIB/LIBSX3.A
Total    282701                   16 files

  終了しました。
```

以上で SX31KIT のインストールは終了です。

callno header のインストール

まず、callno header のインストール先を確認します。

```
  それでは「CALLNO インラインヘッダ」をインストールします。

(確認) インストール先は A:/usr/lang でよいですか?
       よければ "y"、別のパスにする場合は "n" を入力して下さい。

(回答) --->
```

本書付属のインストーラは、3 つのパッケージを同じディレクトリにイ

ンストールすることを前提にしていますので、ここでは原則として「**y**」を入力してください。ここで「**n**」と入力した場合、もしくは callno header のインストールを行わない場合、インストール先のディレクトリを入力する必要があります。詳しくは「SX31KIT のインストール」(p.38)を参照してください。

もし、インストール先の空きディスク容量が足りない場合、インストーラは次のようなメッセージを出力して実行を中断します。空き容量を再確認したのち、最初からインストールをやり直してください。

```
(警告) 指定したディレクトリには「CALLNO インラインヘッダ」を
       インストールするだけの空き容量がありません。

       もう一度確認して最初からインストールをやり直してください。
```

callno header のインストールには、別途作業領域が必要になります。環境変数 temp に、作業領域として 200K バイト程度の空き容量があるディレクトリを設定します。作業領域の空き容量が足りない場合、インストーラは次のようなメッセージを出力して実行を中断します。空き容量を再確認したのち、最初からインストールをやり直してください。

```
(警告) テンポラリディレクトリに「CALLNO インラインヘッダ」を
       インストールするための作業領域が確保できません。

       テンポラリに 200k 以上の空き容量を確保してから、
       インストールをやり直してください。

C:¥>
```

次に Workroom および SX31KIT が正しくインストールされているかどうかを確認します。もし、正しくインストールされていない場合、インストーラは次のようなメッセージを出力して実行を中断します。空き容量を再確認したのち、最初からインストールをやり直してください。

```
Workroom および SX31KIT のヘッダを確認します。

(警告) Workroom および SX31KIT が正しくインストールされていません。
       このままですと、Workroom および SX31KIT からは CALLNO を利用
       できません。
```

ヘッダの確認が終了すると、callno header のインストールを始めます。

```
それでは「CALLNO インラインヘッダ」をインストールします。

テンポラリに展開します。
インラインヘッダをインストールします。
Workroom および SX31KIT のヘッダにパッチをあてます。
          :
         中略
          :
終了しました。
```

これで、callno header のインストールは終了です[4]。

4) CONFIG.SYS中のFILES の値が小さいと、callno header のインストールの途中で止まることがあります。その場合は FILES の値を大きくした後、インストールをやり直してください。

LIBSXC のインストール

まず、LIBSXC のインストール先を確認します。

```
それでは「LIBSXC ライブラリパッケージ」をインストールします。

(確認) インストール先は A:/usr/lang でよいですか？
       よければ "y"、別のパスにする場合は "n" を入力して下さい。

(回答) --->
```

本書付属のインストーラは、3 つのパッケージを同じディレクトリにインストールすることを前提にしていますので、ここでは原則として「**y**」を入力してください。ここで「**n**」と入力した場合、もしくは LIBSXC のインストールを行わない場合、インストール先のディレクトリを入力する必要があります。詳しくは「SX31KIT のインストール」(p.38) を参照してください。

もし、インストール先の空きディスク容量が足りない場合、インストーラは次のようなメッセージを出力して実行を中断します。空き容量を再確認したのち、最初からインストールをやり直してください。

```
(警告) 指定したディレクトリには「LIBSXC ライブラリパッケージ」を
       インストールするだけの空き容量がありません。

       もう一度確認して最初からインストールをやり直してください。
```

空き容量の確認が終了すると、LIBSXC のインストールを始めます。

```
 それでは「LIBSXC ライブラリパッケージ」をインストールします。

 Mode      Size      Date      Time Name
-arw-     36394 94-11-28 01:31:28 A:/usr/lang/CHANGELOG
-arw-      1453 94-12-15 19:44:10 A:/usr/lang/ChangeLog.SX
-arw-      1970 94-11-27 22:00:56 A:/usr/lang/README
d-rwx         0 94-05-02 21:38:04 A:/usr/lang/include/
d-rwx         0 94-05-02 21:41:18 A:/usr/lang/include/sys/
-arw-       788 93-10-07 01:11:08 A:/usr/lang/include/sys/dir.h
-arw-      7293 94-12-15 15:18:54 A:/usr/lang/include/sys/dos.h
                 :
                 :
                中略
                 :
                 :
-arw-      1732 94-11-28 00:56:58 A:/usr/lang/lib/libprof.a
-arw-      3960 94-11-28 00:57:08 A:/usr/lang/lib/libscsi.a
-arw-     11596 94-11-28 00:58:26 A:/usr/lang/lib/libsignal.a
-arw-       132 94-11-28 00:58:32 A:/usr/lang/lib/libsuper.a
-arw-      2872 94-11-28 00:58:38 A:/usr/lang/lib/libtz.a
-arw-      4720 94-11-28 00:58:54 A:/usr/lang/lib/libw.a
-arw-    144622 94-12-15 18:44:36 A:/usr/lang/lib/libsxc.a
Total    752037                   118 files
```

LIBSXC と Workroom を同時に利用する場合、Workroom の SXLIB.L からいくつかのオブジェクトファイルを削除しなければなりません。削除作業は、インストーラが自動的に行います。

```
LIBSXC で Workroom(SXLIB) を利用する場合、そのままではリンク時に
シンボルの衝突が起きます。 これを防ぐため、以下のオブジェクトを
SXLIB から削除します。
      _OPEN.o  _CLOSE.o  _SXKCOMM.o  _SXOBJC.o
```

以上で LIBSXC のインストールは終了です。

ライブラリ形式の選択

本書の推奨開発環境では、すべてのライブラリファイルを AR 形式 (.A) に統一しています。もし、あなたが XC に付属している “LIB.X” を持っているならば、これらをより高速にリンクすることが可能な LIB 形式 (.L) に変換することができます。

ただし、LIBSXC がバージョンアップなどによって再配布される場合、つねに AR 形式で行われますので、なるべく AR 形式のまま使用したほうがよいでしょう。

```
(質問) ライブラリを AR 形式 (.A) から、高速な LIB 形式 (.L) に変換します
か？
      変換するなら "y" を、変換しないなら "n" を入力してください。

      中断するには CTRL+C を押してください。

(回答) --->
```

「y」を選択すると、インストーラはあなたの環境から "LIB.X" を検索し、ライブラリを LIB 形式に変換します。もし "LIB.X" が見つからない場合、インストーラは次のようなメッセージを表示し、ライブラリ形式の変更を行いません。

```
(警告) LIB.X が見つかりません。
       ライブラリは AR 形式 (.A) のままです。
```

"LIB.X" の検索が正常に終了した場合、インストーラはすべてのライブラリファイルを LIB 形式に変換します。

```
ライブラリを LIB 形式 (.L) に変換します。

変換 libsx.a --> libsx.l
変換 libsx3.a --> libsx3.l
          :
          :
         中略
          :
          :
変換 libw.a --> libw.l
変換 libsxc.a --> libsxc.l

終了しました。
```

ライブラリ名の選択

本書の推奨開発環境では、ライブラリファイルの名称が "libc.a" のように XC のライブラリファイル "clib.a" と前後が逆になっています。X680x0 gcc でのみ使用する場合はこのままでもかまいませんが、XC からも利用するのであれば、名称を XC 形式に変更しなければなりません。

```
(質問) ライブラリの名称を XC 形式に変更しますか？
       "clib.a" という XC 形式のライブラリ名を使用するなら "y" を
       "libc.a" という LIBC 形式のライブラリ名を使用するなら "n" を
       入力してください。

       中断するには CTRL+C を押してください。

(回答) --->
```

もし、XC 形式へ変更するならば「**y**」を、このままでよければ「**n**」を入力してください。

なお、ライブラリを LIB 形式に変更した場合、これ以降の画面の表示は“.a”ではなく、“.l”になります。文中の表示例を適宜読み換えてください。

```
ライブラリの名称を XC 形式に変更します。

変換 libsx.a --> sxlib.a
変換 libsx3.a --> sx3lib.a
             :
             :
            中略
             :
             :
変換 libw.a --> wlib.a
変換 libsxc.a --> sxclib.a

終了しました。
```

● AUTOEXEC.BAT の書き換え

ここまででライブラリパッケージのインストールは終了しました。

最後に、ここまでのインストール結果にあわせて環境変数を設定するために、あなたのシステムの“AUTOEXEC.BAT”を修正します。ただし、インストーラは設定例を出力するのみで、実際の修正はあなたが手動で行ってください。

```
  最後にここまでのインストール結果に合わせて環境設定を行います。

A:/usr/lang に AUTOEXEC.PLS の名前で設定すべき環境変数とその値等を
記したファイルを出力します。その内容に従ってあなたの AUTOEXEC.BAT
（あるいはそれに相当するもの）を編集して下さい。
```

インストール先のディレクトリに“AUTOEXEC.PLS”というファイル名

で、必要な環境変数の一覧を出力します。これを参考にして“AUTOEXEC.BAT”をエディタなどで修正してください。“AUTOEXEC.PLS”の出力例を次に示します。

```
REM 以下の環境変数が設定されるように、あなたの AUTOEXEC.BAT を
REM 編集して下さい。
set include=A:¥usr¥lang¥include
set lib=A:¥usr¥lang¥lib
set GCC_LIB=.a
REM 環境変数 GCC_NO_XCLIB が設定されないことを必ず確認して下さい。
```

“REM”で始まる行は注釈ですので、削除してかまいません。ただし、注釈に書かれている内容をよく理解し、確認してから削除するようにしてください。

特に、環境変数 GCC_NO_XCLIB に関する注釈は重要です。GCC_NO_XCLIB は、他の環境変数と違って、変数の内容ではなく、「設定されているか否か」が問題になります。「設定されないことを必ず確認して下さい」と注釈にある場合には、必ず GCC_NO_XCLIB が設定されないことを確認してください。

他の環境変数(include, lib, GCC_LIB)については、AUTOEXEC.BAT 内の別の場所で内容が上書きされることのないように注意してください。確実な方法としては、これらの環境変数の設定を AUTOEXEC.BAT の一番最後で行うとよいでしょう。なお、“AUTOEXEC.PLS”の出力は“COMMAND.X”をシェルとして使用している場合を想定したものです。“FISH.X”や“KSH.X”をお使いの方は、各自で適切な書式に変更してください。

```
インストールはこれですべて終わりです。おつかれさまでした。

C:¥>
```

以上で、推奨開発環境に必要なすべてのパッケージのインストールは終了です。

LIBSXCソースコードパッケージのインストール

ここでは、「LIBSXC ソースコードパッケージ」のインストールについて説明します。ソースコードパッケージをインストールしない場合は、これ以降 50 ページまで読む必要はありません。

特別な環境設定

「必要な環境の準備」(p.36) でも述べたように、ソースコードパッケージを正しくインストールするためには、ファイル名を 21 文字まで正しく認識させるよう、あらかじめ環境設定を行わなければなりません。

Human68k は、ファイル名を 21 文字まで扱えるにもかかわらず、標準ではそのうち先頭の 8 文字までしか実際には認識しません。そのため、9 文字目以降が異なっていても先頭の 8 文字が同じであれば、同じファイルであると誤認識してしまいます。ソースコードパッケージ中のファイルは長い名前が多いため、そのままではいくつかのファイルが上記の理由で上書きされてしまいます。

この問題を解決するためには、“TwentyOne.X” というフリーソフトウェアを利用します。本ソフトウェアは添付 FD に収録されています。ソースコードパッケージ専用のインストーラは、自動的にあなたの環境 (Human68k のバージョンなど) をチェックし、TwentyOne.X を適切に組み込むための手助けをします。もちろん、すでに組み込まれている場合には、この処理は行われません。

インストーラの起動

「インストーラの起動」(p.37) の場合と同様にしてインストーラを起動します。ただし、“INSTALL.BAT” のかわりに “INSTSRC.BAT” を起動してください。

```
C:¥> INSTSRC.BAT
```

インストーラが正常に起動すると、まず “TwentyOne.X” の常駐チェック[5]を行います。

5) TwentyOne.X は動作時にメモリに常駐します。

```
************************************************************
*        LIBSXC Source Code Package - Install Script       *
************************************************************

今から「LIBSXC ソースコードパッケージ」のインストールを行ないます。

"TwentyOne.X" のチェックをします。
```

“TwentyOne.X” が正しく常駐していた場合、「インストール先の指定」(p.48) に進んでください。“TwentyOne.X” が組み込まれていない場合、

次のようなメッセージが表示されますので、“TwentyOne.X” をインストールするディレクトリ名を入力してください。

```
(確認) あなたの環境には "TwentyOne.X" が使用されていません。
       ソースコードパッケージを正しくインストールするには、この
       プログラムをインストールする必要があります。

(質問) "TwentyOne.X" をどのディレクトリにインストールしますか？
       インストール先のディレクトリ名をフルパスで入力して下さい。
       ただしパスの最後に ¥ や / を付けないで下さい。

       中断するには CTRL+C を押してください。

(回答) --->
```

たとえば、A ドライブの “¥sys” というディレクトリにインストールする場合は、次のように入力します。

```
(回答) ---> A:¥sys
```

インストーラは指定されたディレクトリに “TwentyOne.X” をコピーしたのち、次のようなメッセージを表示して実行を中断します。

```
あなたの CONFIG.SYS に

    DEVICE = A:¥sys¥TwentyOne.X +TS

の 1 行を加えたのち、リセットボタンを押して再起動して下さい。

正常に起動したら、もう一度最初からソースコードパッケージの
インストールをやり直して下さい。

C:¥>
```

メッセージに従って CONFIG.SYS を書き換えたのち、リセットしてからもう一度インストーラを起動してください。

インストール先の指定

ソースコードをインストールするディレクトリ名を入力します。

```
（質問）ソースコードをどこにインストールしますか？
        インストール先のディレクトリ名をフルパスで入力して下さい。
        ただしパスの最後に ¥ や / を付けないで下さい。

        中断するには CTRL+C を押してください。

（回答）--->
```

たとえば、A ドライブの“¥usr¥lang”というディレクトリにインストールする場合は、次のように入力します。ディレクトリ名の最後に“¥”や“/”といったパス区切り記号をつけないようにしてください。

なお、LIBSXC のライブラリパッケージとソースコードパッケージは別々のディレクトリにインストールすることも可能ですが、極力、同じディレクトリにインストールするようにしてください。

```
（回答）---> A:¥usr¥lang
```

もし、インストール先の空きディスク容量が足りない場合、インストーラは次のようなメッセージを出力して実行を中断します。空き容量を再確認したのち、最初からインストールをやり直してください。

```
（警告）指定したディレクトリには「LIBSXC ソースコードパッケージ」を
        インストールするだけの空き容量がありません。

        もう一度確認して最初からインストールをやり直してください。

C:¥>
```

空き容量の確認が終了すると、ソースコードのインストールを始めます。環境にもよりますが、すべてをインストールするには数分を要します。

```
    それでは「LIBSXC ソースコードパッケージ」をインストールします。

 Mode      Size      Date      Time Name
-arw-     36394 94-11-28 01:31:28 a:/usr/lang/CHANGELOG
-arw-      1453 94-12-15 19:44:10 a:/usr/lang/ChangeLog.SX
-arw-      1970 94-11-27 22:00:56 a:/usr/lang/README
                 :
                 :
               中略
                 :
                 :
-arw-      1242 94-11-26 23:50:16 a:/usr/lang/src/DefaultRules
-arw-      1549 94-11-26 23:50:24 a:/usr/lang/src/Makefile
-arw-      1497 94-12-15 15:50:00 a:/usr/lang/src/Makefile.SX
Total  1230613                    1069 files

終了しました。

C:¥>
```

以上でLIBSXCソースコードパッケージのインストールは終了です。

添付CD-ROMからのインストール

本節では、本書の添付CD-ROMに収められたフリーソフトウェア(以下、フリーソフト)を、SX-WINDOW上で動作するインストールプログラムを使ってハードディスク等にインストールする方法について説明します。

添付CD-ROMについて

はじめに、添付CD-ROMの内容と取り扱い上の注意点について簡単にふれておきます。

収録内容

本書の添付CD-ROMには、添付FDにも収録した「SX-WINDOW ver.3.1開発キット」のほかに、現在大手BBSなどで公開されているSX-WINDOW対応フリーソフト(一部シェアウェア[1]も含まれる)のほとんどと、Cコンパイラなど SX-WINDOW アプリケーション開発に必要と思われる Human68k 対応フリーソフトを厳選して収録いたしました。

特に SX-WINDOW 用フリーソフトについては、ソフトウェア作者の方々のご協力を仰ぎ、可能なかぎりソースファイルも含めて収録いたしました。これから SX-WINDOW のアプリケーション開発を始めようという人のよき指針として活用していただければ幸いです。

また、添付CD-ROMはISO9660レベル1規格[2]に従ったフォーマットで作成されており、X680x0シリーズとCD-ROMドライブを用意していただければ、誰でも利用することができます[3]。

1) ソフトウェアを継続して使う場合には定められた金額を作者に寄付する必要があります。

2) CD-ROM の記録方法やファイルシステムとしてのフォーマットを定めた規格の1つ。レベル1では、ファイル名に使える文字は "A-Z、0-9、_(アンダーバー)" のみと規定されています。

3) CD-ROM デバイスドライバをお持ちでない場合は、後述のインストール方法に従って添付FDに収録されているデバイスドライバを組み込んでからご利用ください。

使用許諾条件

添付CD-ROMは、以下の使用許諾条件に従ってご利用ください。

○ 添付CD-ROMに収録されているフリーソフトは、それぞれのソフトウェア作者ならびに提供元がその著作権を有しています。各ソフトウェアの使用許諾条件、再配布条件については、ソフトウェアに付属するドキュメント類に記載されていますので、ご使用になる前に必ずお読みください。特に、フリーでないソフトウェア[4)]については、その使用に制限がありますので、提供元が定める使用許諾事項に注意してください。

○ 添付CD-ROMに収録されているソフトウェアについて、作者はもとより、ソフトバンクならびに筆者らは、ソフトウェアの品質、機能および性能についてなんら保証するものではありません。筆者らは、これらのソフトウェアの収録に際して簡単な動作確認を行っていますが、動作の結果に対していかなる種類の保証もいたしません。

4) シェアウェアやSX31 KIT, callno header, 添付FDのなかでは計測技研提供のCD-ROMデバイスドライバがこれにあたります。

収録ソフトウェア

添付CD-ROMに収録されたソフトウェアは、ソフトウェア作者の方々のご厚意により収録を許可していただいたものです。ソフトウェアによっては開発途中のものもありますが、今後のバージョンアップなどのサポートについて筆者らならびにソフトバンクはいっさい関与しません。バージョンアップや質問については、APPENDIX C章「SX-WINDOWフリーソフト一覧」(p.171)にサポートネットとして明記されているパソコン通信ネットワークにおいて各自で対応していただくことになります。

以上の注意事項を守っていただいたうえで、添付CD-ROMを大いに活用してください。

それでは、添付CD-ROMに収められたソフトウェアのインストール方法について順次説明しましょう。

CD-ROMデバイスドライバのインストール

CD-ROMデバイスドライバ

Human68k ver.3.02は、標準ではCD-ROMデバイスがサポートされ

ていません。そのため、本書添付の CD-ROM にアクセスするためには、サードパーティの製品やフリーソフトとして公開されている CD-ROM デバイスドライバを別途組み込む必要があります。

本書の添付 FD には、1.2 (株) 計測技研のご厚意により提供していただいた機能限定版 CD-ROM デバイスドライバ[5]"cddev.sys" と、後藤浩昭 (GORRY) 氏[6]作のフリーの多機能 SCSI デバイスドライバ SUSIE[7]を収録しています。

本書のインストール解説では、計測技研提供の機能限定版の cddev.sys の組み込みを前提としていますが、cddev.sys を組み込んでも CD-ROM ドライブが認識されない場合は、後者の SUSIE を組み込んでからお試しください[8]。いずれのデバイスドライバを組み込んでも認識できない場合は誠に申し訳ありませんが、添付 CD-ROM のご利用をあきらめていただくこととなります[9]。

5) 製品版の売りのひとつであるオーディオコマンドの機能が省略されています。

6) ディスク内容の整理をすることでアクセス速度を向上させる "refreshg.x" や、グラフィック表示プログラム apicg.r などの作者。

7) SUSIE は、SCSI デバイスのための多機能デバイスドライバで、1995年7月13日時点での最新版は SSE_107A.LZH です。

8) SUSIE では、PC-98 や IBM PC/AT 互換機、FM TOWNS 用の CD-ROM ドライブも認識できるようですが、どんな CD-ROM ドライブでも必ず認識できるわけではない点に注意してください。

9) もし友人に IBM PC/AT 互換機や PC-98、FM TOWNS をお持ちの方がおられたら、それらのマシンに CD-ROM ドライブを接続してファイルを取り出すこともできます。

CD-ROM ドライブの接続

まず、デバイスドライバをインストールする前に、CD-ROM ドライブが X680x0 と接続されていることを確かめてください (X68000 SUPER 以降の機種では SCSI 端子同士を SCSI ケーブルで接続します)。

X68000 SUPER 以前の SCSI 端子がない機種をお使いの方は、SCSI カードを購入するか、フリーソフトの SxSI[10]を組み込んだうえで、添付 FD に収録した前述の SUSIE を CONFIG.SYS に組み込んでください[11]。

CD-ROM ドライブと X680x0 本体との接続を確認したら、CD-ROM ドライブの電源を入れ、次に本体の電源を入れて X680x0 を起動してください。このとき、すでに添付の CD-ROM をセットしておいてもかまいません。

10) SASI 端子しかない機種で SCSI ハードディスクや CD-ROM を接続できるようにするドライバ。

11) 組み込み方法の詳細については、各ソフトウェアに付属のドキュメントを参照してください。

デバイスドライバの登録

次に、X680x0 の起動時に CD-ROM ドライブが認識され、アクセス可能となるよう、前述の CD-ROM デバイスドライバを CONFIG.SYS に登録します。

すでに、なんらかの CD-ROM デバイスドライバを CONFIG.SYS に登録されている方は、本項での作業は不要ですので次項に進んでください。

以下では、説明の都合上、「2.1　添付 FD からのインストール」 (p.34)

で作成したインストールディスク[12]を挿入したドライブをCドライブとし、デバイスドライバを登録するCONFIG.SYSのある起動ドライブをAドライブとします。また、作業ディレクトリを「A:¥TMP」、Human68kのデバイスドライバなどのシステムファイルが入っているディレクトリを「A:¥SYS」として話を進めます。

12)安全のため、添付ディスクをバックアップした作業用ディスクのこと。

インストールディスクをX680x0本体のフロッピーディスクドライブの0番に挿入し、インストールディスクを挿入したドライブ(ここではCドライブ)にカレントドライブを変更してください。そして、インストールディスクに含まれているファイル“CDROM21.LZH”を作業ディレクトリにコピーします。

```
A:¥> C:
C:¥> COPY CDROM21.LZH A:¥TMP
```

次に、カレントディレクトリを作業ディレクトリに変更します。

```
C:¥> A:
A:¥> CD TMP
```

そして、添付FDの¥ETCディレクトリに収められているlha_x647.xをカレントディレクトリにコピーし、実行します(lha_x647.xは、アーカイブユーティリティLHaの自己解凍型ファイルです)。すると、アーカイブファイルが順に展開されていきます。その中のlha.xを使って、先ほどコピーしたファイル“CDROM21.LZH”の中身を展開します。

```
A:¥TMP> LHA E CDROM21.LZH

Extracting from Archive : keisoku/CDROM21.LZH

cddev.sys              : 解凍終了    [    11874 ] (    11874 )
readme.doc             : 解凍終了    [     2068 ] (     2068 )
```

計測技研提供のcddev.sysを使う場合には、展開されたファイルの1つreadme.docにCD-ROMデバイスドライバの使用条件[13]が記載されていますので、必ず目を通してください。

13)対応CD-ROMドライブについても記載されています。

上記のように圧縮ファイルの展開が終わったら、CD-ROMデバイスドライバ本体cddev.sysをシステムファイル・ディレクトリA:¥SYSにコピーします。

```
A:¥TMP> COPY cddev.sys ¥SYS
```

⊙コラム：cddev.sys の起動オプション

cddev.sys を利用する場合、基本的にオプションを指定する必要はありません。オプションを指定しない状態で CD-ROM ドライブが認識されない場合は、下記のように “ID” オプションによって、CD-ROM ドライブの SCSI ID 番号を指定してみてください。

DEVICE=A:¥SYS¥cddev.sys /ID5
(CD-ROM の ID が 5 の場合)

オプションの詳細は、アーカイブファイル “CDROM21.LZH” に含まれているドキュメント “README.DOC” を参照してください。

コピーが終了したら、CONFIG.SYS に次の 1 行をエディタなどで追加して、X680x0 の起動時にインストールした CD-ROM デバイスドライバが登録されるようにしてください。

List 2-1 ● CD-ROM ドライバの組み込み

```
1:  DEVICE = A:¥SYS¥cddev.sys
```

上記のように書き加えたあと、ファイル CONFIG.SYS をセーブしてください。エディタを終了し、TYPE コマンドなどで CONFIG.SYS が正しく書き換えられていることを確認したら、リセットボタンを押して X680x0 を再起動してください。

X680x0 の再起動が完了したら、CD-ROM ドライブが正しく認識されているかどうか、次のように DRIVE コマンドを使って確認します。

```
A:¥TMP> DRIVE

A: ハードディスク (SCSI) ユニット番号....   0
B: 光磁気ディスク (SCSI) ユニット番号....   0
C: 2HD (1MB)             ユニット番号....   0
D: 2HD (1MB)             ユニット番号....   1
E: CDROM (SCSI)          ユニット番号....   0
F: 仮想ドライブです
|
Z: 仮想ドライブです

A:¥TMP>
```

上記のように、CD-ROM ドライブが正しく認識されれば、本項の目的である CD-ROM デバイスドライバのインストールは終了です。次は、添付 CD-ROM に収録されているソフトウェアのインストールプログラム[14]を利用するための設定を行います[15]。

14) 本プログラムは SX-WINDOW 上で動作するため、あらかじめ SX-WINDOW が利用できる環境を用意しておく必要があります。

15) 計測技研提供の cddev.sys は、メディアが挿入されていない MO ドライブを CD-ROM ドライブと誤って認識してしまう場合もありますので、注意が必要です。

⊙コラム：計測技研の CD-ROM ドライバ対応 CD-ROM ドライブ

添付 FD に収録されている CD-ROM デバイスドライバ `cddev.sys` が対応している CD-ROM ドライブは次のとおりです。

○ SCSI に対応している CD-ROM ドライブ (ただし、NEC PC-CD10, PC-CD30, Apple CD150 を除く)

`cddev.sys` の開発元である計測技研で動作を確認している CD-ROM ドライブは以下のとおりです (1995 年 4 月現在)。

○ TOSHIBA XM3301, XM3401, XM 4101, XM5901
○ SONY CDU-561, CDU-55S (Ver. 1.0f, Ver.1.0q で確認)
○ PLEXTOR 製のドライブ
○ 松下寿製のドライブ
○ NAKAMICHI MBR-7
○ロジテック LCD-440
○ DATA WEST DWR-22MS

ここに記載されている CD-ROM ドライブは、あくまでも計測技研で動作を確認したドライブです。これ以外の CD-ROM ドライブでも動作する可能性はありますが、保証のかぎりではありません。また、CD-ROM ドライブが認識されないからといって、提供元の計測技研に質問などをしないようにしてください。

添付 CD-ROM の内容

添付 CD-ROM のディレクトリ構成は Table 2-2のようになっています。

Table 2-2 ● 添付 CD-ROM のディレクトリ構成

ディレクトリ名	分類項目
APL	アプリケーション
ARC	アーカイバアプリケーション
DATA	リソースデータ
DESKTOP	デスクトップアプリケーション
DEVELOP	開発アプリケーション
DOC	ドキュメントデータ
EDITOR	エディタアプリケーション
EXT	SX-WINDOW を拡張するアプリケーション
FILE	ファイルアプリケーション
GAME	ゲーム
GRAPH	グラフィックアプリケーション
LANG	言語アプリケーション
LIB	開発用ライブラリ
MUSIC	FM 音源、ADPCM 音源アプリケーション
SX110	SX-WINDOW ver.1.10 専用アプリケーション
TERM	通信アプリケーション
TOOL	ツールアプリケーション

それぞれのアプリケーションプログラムは、Table 2-2のようなアプリ

ケーションのジャンルごとに分類して収められています。各ジャンル名のディレクトリの下にはアプリケーションごとに分けられたディレクトリがあり、その下に展開されたアプリケーションプログラムと、アーカイブファイルを収めた¥ARCというディレクトリがあります。

以下に、例として通信ソフトQuTERMのディレクトリ構成を示します。

```
¥TERM                                 : 通信アプリケーションのディレクトリ
 └──QUTERM                            : QuTERMのディレクトリ
      ├──ARC                          : アーカイブのディレクトリ
      │   ├──QUTERM54.LZH             : アーカイブファイル
      │   └──QUTRM54S.LZH             : ソースプログラムのアーカイブファイル
      ├──QUTERM.X                     : アーカイブから展開されたファイル
      ├──QUTERM.HLP
      └──QUTERM.LB
            :
```

前述のとおり、添付CD-ROMはISO9660レベル1規格に従ったフォーマットで作成されているため、CD-ROMに収録したアプリケーションのうち、ISO9660レベル1規格では使えない文字列からなるファイル名(漢字やカタカナなど)のものは変更させていただきました[16)]。添付CD-ROMには、主なアプリケーションの実行に必要なファイルとドキュメントファイルを展開した状態で収録しましたが、上記の理由により添付CD-ROMに収録されている実行ファイルを直接実行することはできません。

CD-ROM内のアプリケーションソフトをインストールするには、SX-WINDOW上で動作するインストールプログラム(**インストーラ**.X)を使う方法[17)]か、¥ARCディレクトリに含まれているアーカイブファイルからユーザ自身が手作業で展開する方法の2通りがあります。

本節では、一部ソフトウェアのインストールに必要な“TwentyOne.X”について簡単に説明したあと、インストーラを使ってハードディスクにインストールする方法と、手作業でハードディスクにインストールする方法を順を追って解説します。

16) 当然ながら、大文字小文字の区別もありません。

17) インストーラを使うことによって、ハードディスクへアプリケーションソフトをインストールする際にオリジナルのファイル名に自動的に戻します。

TwentyOne.Xの常駐

Human68kでは、最大で21文字のファイル名が使えるような機構があるものの、実際にはそのうち8文字(+拡張子3文字)しか認識されないという制限があります[18)]。TwentyOne.Xは、この制限を取り払うことによって21文字すべてを認識することができるようにし、さらに大文

18) たとえばSXWINDOW.XとSXWINDOWS.Xは同じファイルと見なされてしまいます。

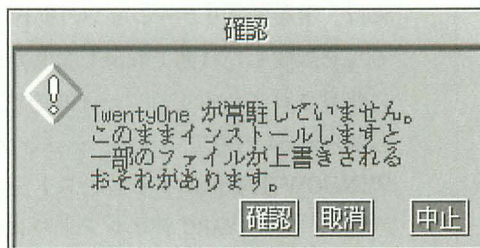
字・小文字を区別したり、ファイル名の中に複数のピリオドが使えるようにHuman68kを拡張する常駐ソフトウェアです。X680x0用のアプリケーションソフトウェアには、このTwentyOne.Xが常駐していることを前提とした、8文字以上の長いファイル名のものや、複数のピリオドを使ったものが多数存在しています。

添付CD-ROMに含まれているアプリケーションのうち、「エディタアプリケーション」と「開発アプリケーション」、「開発用ライブラリ」には、上記のような特殊なファイル名を用いたものが含まれています[19)]。これらをインストールする場合には、前述のTwentyOne.Xの機能が必要となりますので、インストーラを実行する前にTwentyOne.Xを常駐させておく必要があります。

19) たとえば、String.hとstring.hが同じディレクトリに含まれているなど。TwentyOne.Xを常駐させていないHuman68kではどちらも同じファイル名として認識されます。

TwentyOne.Xのインストール方法については「インストーラの起動」(p.47)を参照してください。

TwentyOne.Xが常駐していない状態でインストーラを実行すると、Fig. 2-1のような警告メッセージが表示されます。

Fig. 2-1 ● TwentyOne.Xが常駐していない場合に出る警告メッセージ

このような警告メッセージが表示された場合は、確認ボタンをクリックすれば警告メッセージが消えて、インストーラが起動します。この状態でインストール作業を続行することは可能ですが、一部のアプリケーションを正しくインストールすることができません。一度SX-WINDOWを終了してTwentyOne.Xを常駐させたうえで、再度SX-WINDOWを起動してください。

それでは、いよいよアプリケーションのインストールについて説明します。

インストーラを用いたインストール

まず最初に、添付CD-ROMに含まれているインストーラの役割につ

いて説明します。

本書で用意したインストーラは、SX-WINDOW 上で動作するインストーラ.X を用いたもので、主な処理としては、

○ CD-ROM からハードディスクなどへファイルのコピーを行う。

○ ISO9660 レベル 1 規格に従うように変更されたファイル名を元のファイル名に戻す。

が挙げられます。本書のインストーラは、ファイル名を元のファイル名に戻してコピーするだけのプログラムであり、いわゆるアプリケーションをインストール (使えるように) する一般的なインストーラとは違いますので注意してください。

また、アプリケーションによっては、実際に利用 (実行) する前に設定が必要なものもありますが、本書のインストーラはそこまで対応していません。それぞれのアプリケーションのドキュメントを参照して、各自で設定作業を行ってください。ソースプログラムなどは、開発用ライブラリに分類されているアプリケーションを除いてインストールを行いません。ソースプログラムが必要な場合は後述する手動インストールの項目を参照してください。

それでは、インストール作業を順次見ていきましょう。

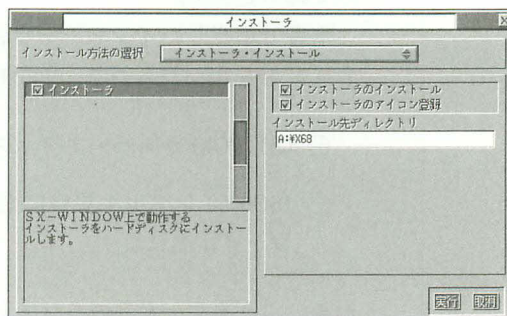
添付 CD-ROM のルートディレクトリには、シャープ (株) の許可を得てインストーラ.X とインストーラ.LB を、ISO9660 レベル 1 規格のファイル名規則にあわせてそれぞれ SETUP.X と SETUP.LB に変更して収録してあります[20)]。CD-ROM 内のファイルのコピーには SETUP.X と SETUP.LB をお使いください。

20) インストーラ.X、インストーラ.LB は全角カタカナのファイル名ですので、ISO9660 のファイル名として使えるように変更させていただきました。

まず、SX-WINDOW を起動して CD-ROM ドライブのアイコンをダブルクリックし、ディレクトリを表示します。

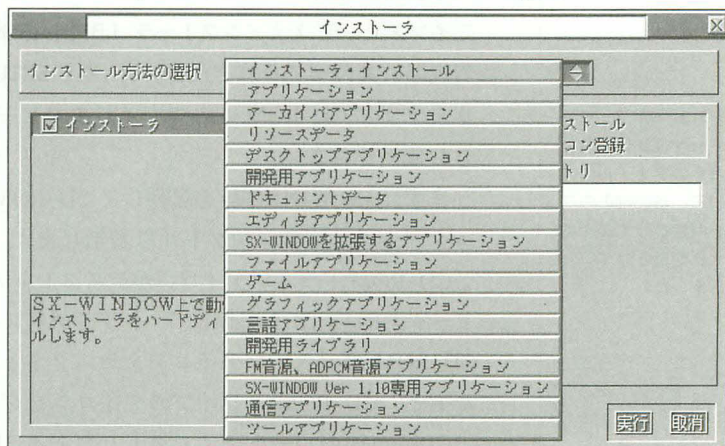
CD-ROM ドライブアイコンがデスクトップに表示されていない場合は、ドライブトレイ.X を実行して、ドライブトレイのウィンドウの内部にある CD-ROM のドライブアイコンをデスクトップにドラッグ & ドロップしたあと、CD-ROM ドライブのアイコンをダブルクリックしてください。

CD-ROM のルートディレクトリにある SETUP.X をダブルクリックすると、インストーラが起動し、Fig. 2-2 のようなウィンドウが表示されます。

Fig. 2-2 ● 起動直後のインストーラ画面

インストーラのウィンドウ画面には、次のような項目が表示されています。

○インストール方法の選択

インストーラウィンドウの上方にある「インストール方法の選択」の右側には、横に長いボタンがあります。このボタンには、現在選択中のアプリケーションソフトの分類が表示されています。このボタンの上でマウスの左ボタンを押し続けると、インストールするアプリケーションの分類項目が Fig. 2-3のように表示されます。このとき、左ドラッグして、インストールしたいアプリケーション分類のうえでマウスの左ボタンを離せば、インストールするアプリケーションの分類を変更することができます。

Fig. 2-3 ● インストールするアプリケーションの分類項目

○インストール概略表示部

インストーラウィンドウの左側にはインストールするアプリケーションのアーカイブ名が表示されます。チェックボックスをクリックする

と、チェックマークがついたり消えたりします。チェックマークのついている項目のみがインストールされますので、すでにインストールされていてあらためてインストールする必要のないアプリケーションをインストールしないように設定することができます。

インストールする内容の概略部分をクリックすると、右のインストール詳細表示部にインストール内容の詳細が、左下のヘルプ表示部には概略が表示されます。インストールする内容がそこに表示しきれない場合には、スクロールバーが表示されますので、操作してスクロールさせることができます。

○インストール詳細表示部

インストーラウィンドウの右側には、インストール概略表示部で選択された項目の詳細な設定内容が表示されます。設定内容は、インストール概略表示部で選択された項目によって異なります。

○ヘルプ表示部

インストーラウィンドウの左下には、インストール概略表示部で選択された項目に関する説明が表示されます。

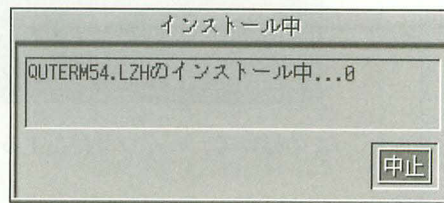
○実行ボタン

実行ボタンをクリックすると、チェックボックスでチェックされた内容に従ってインストールが開始されます。インストール中はFig. 2-4のようなインストールダイアログが表示されます。

Fig. 2-4 ● インストールダイアログ

インストール中

QUTERM54.LZHのインストール中...0

中止

メッセージ表示部には、現在どのアプリケーションをインストールしているのかなどを示すメッセージが表示されます。メッセージの横の数字は、インストールされているファイル数を示しています。インストール中に中止ボタンを押すと、インストールは中止されます。さらに、ここで中止ボタンを押すと、これまでにインストールされたアプリケーション名の前についたチェックマークが消えています。インストールを再開するときは、実行ボタンを押すと、インストールの終わっていないアプリケーションのインストールを始めます。

○取消ボタン

取消ボタンをクリックすると、インストーラが終了します。

インストーラのウィンドウ画面は以上のようになっています。

●インストーラのインストール

それではインストーラのインストールを始めましょう。「インストール方法の選択」の右側にある横に長いボタン部分を左クリックすると、分類項目がFig. 2-5のように表示されますので「インストーラ・インストール」を選択してください。

まず最初に、SETUP.XとSETUP.LBをハードディスクにインストールします。

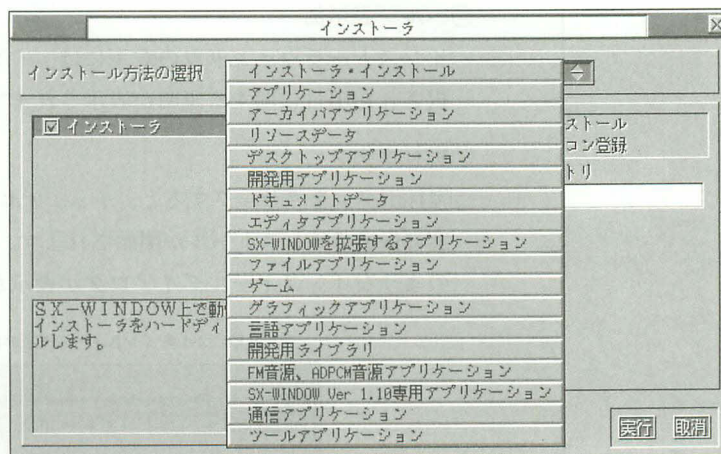
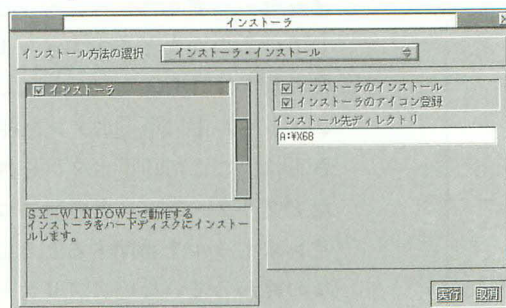
Fig. 2-5 ● 分類項目の画面

この操作で、いくつか存在するインストーラのなかで「インストーラ・

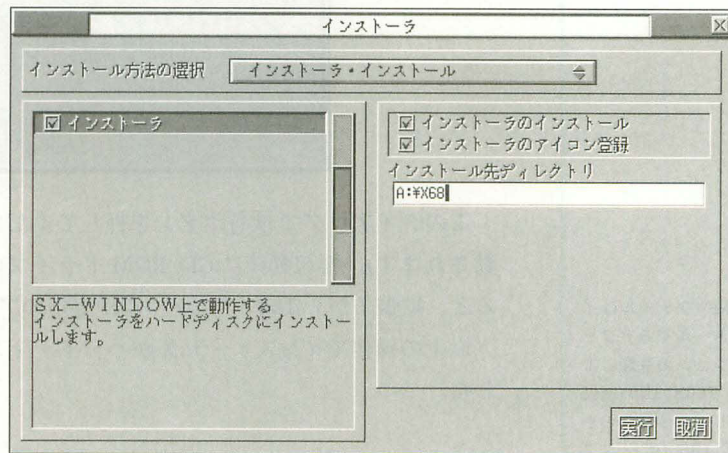
Fig. 2-6 ● インストーラ・インストールの起動画面

インストール」という名前のインストーラが起動します。すでにインストーラ・インストールが起動している場合は何もしなくてかまいません。

インストール詳細表示部と、ヘルプ表示部が Fig. 2-6のように表示されます。

インストール詳細表示部の「インストール先ディレクトリ」の下にある文字列部分[21]を左クリックすると、インストール先ディレクトリをキーボードから入力できるようになりますので、SETUP.X と SETUP.LB をインストールするディレクトリ名を入力してください。

21)デフォルトでは A:¥X68 が表示されています。

設定後の画面を Fig. 2-7に示します。

Fig. 2-7 ● インストーラ・インストール設定後の画面

なお、インストールが終了すると、SETUP.X と SETUP.LB は、元のインストール.X とインストール.LB に名前が変更されています。添付 CD–ROM に収録されている SETUP.X と SETUP.LB は、現在入手できる最新バージョン(タイムスタンプは 95-02-22 12:00:00) のインストーラ.X、インストーラ.LB の名前を変更したものですので、すでに SX–WINDOW ver.3.1 システムキットや、開発キットツール集などでインストーラ.X を入手されている方も、再度 SETUP.X と SETUP.LB をインストールするようにしてください。

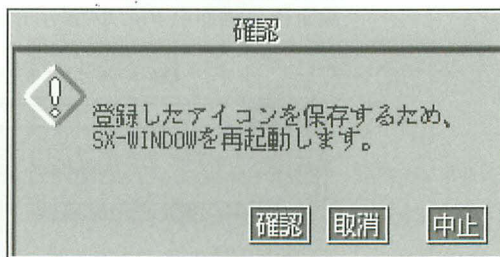
すでに古いバージョンのインストーラ.X がインストールされている場合は、そのファイルがあるディレクトリ名を指定して、インストーラ.X が本書添付の新しいもので上書きされるようにしてください。古いバージョンのインストーラ.X が残っていると、以後のインストール作業が正しく行われる保証はありませんので、ご注意ください。

インストール先ディレクトリの設定が終了したら、実行ボタンを押し

てください。添付CD-ROM上にある最新バージョンのSETUP.Xがインストーラ.Xとしてハードディスクにインストールされます。

「インストーラ・インストール」の内部ではアイコンの定義と設定を行っていますので、この設定を有効にするために一度SX-WINDOWを再起動しなければなりません。

インストールが終了すると、「登録したアイコンを保存するため、SX-WINDOWを再起動します」というダイアログが表示されます (Fig. 2-8)。

Fig. 2-8 ● 再起動のダイアログ

このダイアログで実行ボタンを押してください (SX-WINDOWが再起動されます)。再起動後にCD-ROMドライブのディレクトリを表示させると、拡張子が“.INS”のファイルが専用のアイコンで表示されます[22]。

22) “.INS”ファイルはインストールするアプリケーションの分類によって、TERM.INS (通信アプリケーション) やGAME.INS (ゲームプログラム) などとなっています。

以上の操作でインストーラ.Xがハードディスクにインストールされました。

アプリケーションのインストール

次は、添付CD-ROMに収録されているアプリケーションのインストール方法について説明します。

SX-WINDOW上からCD-ROMドライブのルートディレクトリにあるSETUP.Xをダブルクリックしてインストーラを起動してください。

「インストール方法の選択」の右にあるボタンを左クリックすると、Fig. 2-5のように分類項目が表示されますので、インストールしたいアプリケーションの分類を選択してください。ここでは、例として「通信アプリケーション」を選択します (Fig. 2-9)。

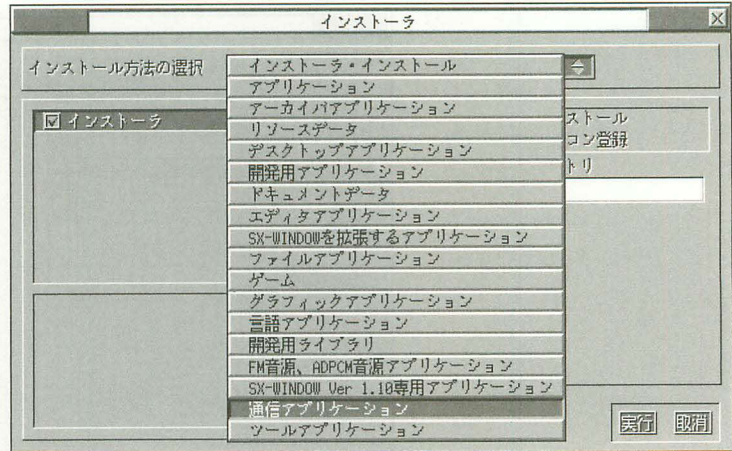
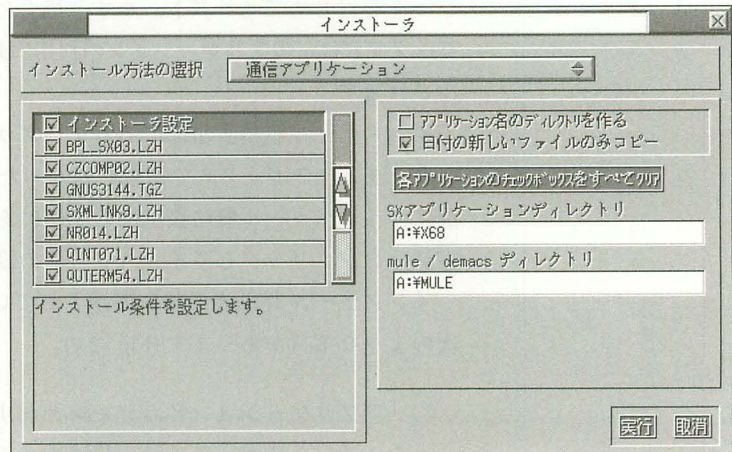
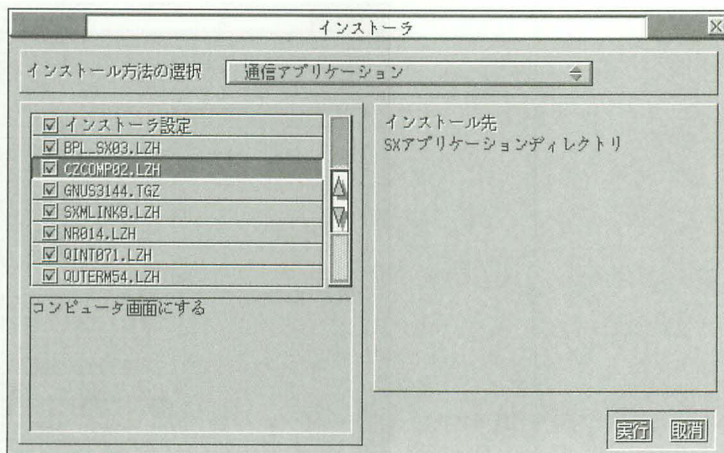
Fig. 2-9 ● 通信アプリケーションのインストーラ画面の選択

「通信アプリケーション」インストーラの起動直後の画面を Fig. 2-10 に示します。

Fig. 2-10 ● 通信アプリケーションのインストーラ起動画面

インストール概略表示部 (インストーラウィンドウの左側) には、アプリケーションのアーカイブ名が表示されています。インストール概略表示部の左側にあるチェックボックスにチェックがついていると、そのアプリケーションがインストールされることを示しています。

インストール概略表示部のアプリケーションのアーカイブ名が表示されている部分をクリックすると、ヘルプ表示部に該当するアプリケーションの簡単な説明が表示されます (Fig. 2-11)。

Fig. 2-11 ● アプリケーションの概略説明

すでに該当するプログラムがインストールされており、新たにアプリケーションをインストールしたくない場合には、インストール概略表示部のチェックボックスを左クリックしてオフにします。そのアプリケーションをはじめてインストールする場合は、そのまま実行ボタンを左クリックすれば、アプリケーションの利用に必要なプログラムがインストールされます。

インストール概略表示部にあるプログラムをインストールするかしないかをチェックボックスで指定し終わったら、インストールを実行する前に、インストール概略表示部の先頭にある「インストーラ設定」の文字部分を左クリックしてインストーラの設定を行います。

インストール詳細表示部に表示されているインストーラの設定には、次のような項目があります (Fig. 2-7)。

○「アプリケーション名のディレクトリを作る」チェックボックス

このチェックボックスは、後述する項目「SX-WINDOW 実行ファイルディレクトリ」で指定されたディレクトリ名 (デフォルトでは A:¥X68) に、アプリケーション名からなるサブディレクトリ (たとえば、¥QUTERM) を作成したあと、そのディレクトリ (たとえば、A:¥X68¥QUTERM) に実行ファイルなどをインストールする指定です。ただし、アプリケーションごとにディレクトリを作成していくと、アプリケーションを利用する段階で SX-WINDOW が実行ファイルを検索する際に時間がかかることなどから、デフォルトでは、この設定を OFF にしています。できるかぎりディレクトリを分割せず、アプリケーションを同じディレクトリにインストールするようにして

ください(詳細は次ページのコラムを参照)。

○「日付の新しいファイルのみコピー」チェックボックス

このチェックボックスは、インストール先ディレクトリにインストールしようとしているファイルと同じものがあった場合に、既存のファイルとCD-ROM内部のファイルの日付とを比較して、CD-ROM内部のファイルの日付が新しい場合にのみコピーする指定を行うオプションです。デフォルトではONになっています。

○「各アプリケーションのチェックボックスをすべてクリア」ボタン

このボタンを押すと、インストール概略表示部に表示されている各アプリケーションのチェックボックスがすべてOFFになり、各アプリケーションのインストール動作を行わない設定になります。

インストール概略表示部にたくさんのアプリケーション項目があるような場合、デフォルトでONになっているすべてのアプリケーションのチェックボックスをひとつひとつOFFにするのが大変な場合に利用します。

○SXアプリケーションディレクトリ

SX-WINDOWアプリケーションをインストールするディレクトリを指定します。

この他の分類のアプリケーション(開発用アプリケーションなど)では、さらに次のような項目が表示されることがありますので、適宜、指定してください。

○Humanアプリケーションディレクトリ

Human68k上で動作するアプリケーションの実行ファイルをインストールするディレクトリを指定します。

○コンパイラ / ライブラリディレクトリ

コンパイラや、専用アセンブラ、各種ライブラリをインストールするディレクトリを指定します。

○mule / demacsディレクトリ」

muleやdemacsをインストールするディレクトリを指定します。

インストール方法の選択に表示されているアプリケーションの分類が「エディタアプリケーション」、「開発用アプリケーション」または「開発用

⊙コラム：実行ファイルの検索

SX–WINDOW では、広大なハードディスク内部に散在する実行ファイルを検索する際に、過去に実行した実行ファイルのあったディレクトリ名を SYSDTOP.SX というリソースファイルのなかに記憶しておき、その記憶をもとにファイルの検索を行います。これは、Human68k の環境変数 PATH に近い機能ともいえますが、SX–WINDOW では実行ファイルを実行するたびに自動的に内容を新しく更新している点で異なります。記憶されているディレクトリ名の場所に実行ファイルが存在しなかった場合は、ハードディスク内のすべてのディレクトリ領域を検索するようになっています。

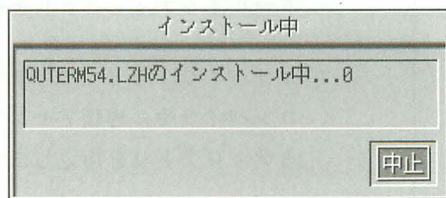
しかし、SYSDTOP.SX には実行ファイルのディレクトリを記憶する領域は 255 バイトしかありませんので、あちこちのディレクトリに実行ファイルが散在していると、実行するたびに新しいディレクトリを記憶することで、過去のディレクトリ記憶が消去されてしまいます。このため、アプリケーションを起動する際に「検索中です」のダイアログが表示され、ハードディスク内部を検索している間、待たされることが多くなります。

このような現象を回避するためには、実行ファイルを特定の 1 つか 2 つかのディレクトリ (たとえば、“¥X68” など) に集めておくことが重要です。

添付 CD–ROM のインストーラにおいて、デフォルトではアプリケーションごとのサブディレクトリを作成しない設定となっている理由は、実行ファイルを 1ヵ所にまとめて置いておくためなのです。

ライブラリ」の場合は、ファイル名の大文字小文字が区別されるモードでインストールされます。このため、上記の項目で設定するディレクトリ名の大文字小文字の違いにも注意しつつ設定してください。

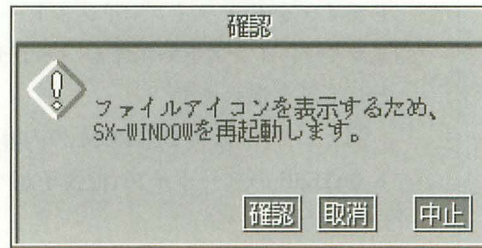
これらすべての設定が終了したら、実行ボタンを押すとインストール作業が始まります。インストール中は Fig.2-12 のようなダイアログが表示されます。

Fig. 2-12 ● インストール中のダイアログ

ダイアログの中に表示されている数字は、現在コピーしているファイルの数が表示されます。ファイル数の少ないアプリケーションでは 2 つ、非常に大きなアプリケーションでは 1000 を超えることがあります。

インストール中に中止ボタンを押すとインストールは中止されます。こ

の場合、すでにインストールを終了したアプリケーションについては、インストーラの概略表示部のチェックマークが消えていますので、インストールを再開したい場合はもう一度実行ボタンを押すことにより、インストールを中断したアプリケーションからインストールを再開することができます。

インストールが終了すると、「ファイルアイコンを表示させるため、SX-WINDOW を再起動します。」というダイアログ（Fig.2-13）が表示されます。

Fig. 2-13 ● ファイルアイコン表示のための再起動ダイアログ

インストーラによってハードディスクにコピーされたファイルは再起動することによって初めて SX-WINDOW で表示されるようになります。これは、SX システムが標準では長いファイル名やマルチピリオドを用いたファイル名を取り扱うことができないため、インストーラではファイルのコピーを Human68k の機能を利用して行っており、インストール直後の状態ではインストールしたファイルのアイコンが表示されないためです。

インストールしたファイルのアイコンを表示させたい場合は実行ボタンを、他の分類のアプリケーションを続けてインストールしたい場合は中止ボタンを押してください。

再起動後に表示されるファイルアイコンが、他のアイコンの上に重なって表示され、見えにくい場合があります。また、SX システムの標準では長いファイル名を持つファイルや、ファイルの先頭にピリオドを持つファイル、複数のピリオドを持つファイルは取り扱えない場合がありますので、上記のようなファイルを取り扱いたい場合には、SX-WINDOW を終了してコマンドラインから取り扱うようにしてください。

前にも述べたように、インストーラが取り扱うのはファイル名の変更とコピーだけですので、アプリケーションによってさらに設定が必要なものは、アプリケーションに付属するドキュメントを参照して必要な設定を行ってください。

以上でインストール(ハードディスクに実行ファイルをコピーする)作業は終了です。

手動インストール

ここでは、付属のインストーラを用いずに手作業でアプリケーションをハードディスクなどにインストールする方法について説明します。

以下では、添付FDを挿入したドライブをCドライブとし、CD-ROMドライブをEドライブ、インストールするハードディスクドライブをAドライブ、インストールするハードディスク上のディレクトリをA:¥X68として話を進めます。

例として、「添付CD-ROMの内容」(p.56)で例として示した通信ソフトQuTERMのアーカイブ(拡張子が".LZH"のアーカイブ)を展開する場合を示します。

```
A:¥>CD X68
A:¥X68>LHA X E:¥TERM¥QUTERM¥ARC¥QUTERM54.LZH
```

アーカイブの拡張子が".TGZ"の場合は、添付FDに収録されているTXF.Rというプログラムを用いて展開します。

TXF.Rを持っていない方は、添付FDに収録されているTXF.Rをハードディスクの中のHuman68kの環境変数PATHで指定されているディレクトリにコピーしておきます。ここでは、A:¥BINにコピーする例を示します。

```
A:¥>COPY C:¥BIN¥TXF.R A:¥BIN
```

TXF.Rを利用して、拡張子が".TGZ"のファイルを展開する方法を次に示します。

```
A:¥>CD X68
A:¥X68>TXF -x E:¥ARC¥VISON¥ARC¥VISON111.TGZ
```

手動インストールでは、上述したようにひとつひとつアーカイブを展開していかなければなりません。

各アプリケーションのソースファイルは、ソフトウェア作者の方々のご協力を得て、可能なかぎり添付CD-ROMにアーカイブの形式で収録

させていただきました。ソースファイルのアーカイブがあるアプリケーションについては、実行ファイルのアーカイブと同様に ¥ARC ディレクトリにソースファイルを収めてあります。

ソースファイルのハードディスクへのコピー作業はインストーラでは扱いませんので、ソースファイルを必要とされる方は、手作業でコピーして展開していただくことになります。

以上、ざっとインストール作業にそって流れを追ってきました。インストール作業そのものは、設定が複雑なだけでそれほど難しいものではありませんので、注意深く設定を行っていただければ、誰にでも簡単にインストール作業を行うことができることでしょう。

本書では、添付 CD–ROM に収められた秀作アプリケーションを読者諸兄のハードディスクにコピーするところまでを手助けいたしますが、それらアプリケーションを使いこなせるかどうかはあなた次第です。添付 CD–ROM に収録した、SXer こと SX–WINDOW 使いらのソフトウェアで、あなたの SX–WINDOW ライフが少しでも華やかになるのであれば嬉しいかぎりです。

最後になりましたが、ソフトウェアの収録を快諾していただけました、数多くのソフトウェア作者の方々に感謝いたします。

第3章

SX-WINDOW ver.3.1開発キット

本章では、添付FDに収録されているSX-WINDOWアプリケーションの開発環境のうち、SX31KITとcallno headerについて説明します。
また、SX-WINDOW ver.3.1で追加された機能の一部についてもサンプルプログラムをまじえて解説をします。

SX31KITとは

C言語でSX-WINDOWのプログラムを開発するには、次のようなものが必要です。

○構造体の定義
SX-WINDOWのさまざまなレコードをC言語から利用するために、構造体として定義しておく必要があります。

○ SXコールの宣言
SXコールをC言語から呼び出すために、あらかじめ関数として宣言しておく必要があります。

○ SXコールのライブラリ
実際にSXコールを呼び出すために必要になります。

通常、前二者はインクルードヘッダとして、後者はライブラリファイルとして準備しておきます。

シャープから発売されているWorkroomには、C言語でSX-WINDOWのプログラムを開発するために必要なインクルードヘッダとライブラリファイルが付属しています。しかし、残念なことに、Workroom付属のインクルードヘッダやライブラリファイルはSX-WINDOW ver.2.0までにしか対応していません。

本書で提供するSX31KITは、SX-WINDOW ver.3.1までに新たに追加された機能をC言語から利用するための開発環境です。SX31KITはWorkroomを拡張する形で構成されています。そのため、

SX31KITの利用にはWorkroomが必要不可欠

です。

SX31KITはXC, gccのどちらでも利用することができます。しかし、第1章でも述べたように、本書ではCコンパイラとしてgccを推奨しています。SX31KITは「3.3　callno header」(p.111)で紹介するcallno headerとともに利用することで大きな力を発揮します。ただし、callno headerを利用するには、コンパイラはgccでなければなりません。

本節では SX31KIT を利用した簡単なサンプルプログラムを通して、SX31KIT の基本的な使い方を説明します。はじめて C 言語で SX-WINDOW のアプリケーションを作成しようという方は、説明をよく読んで基本を学んでください。また、すでに SX-WINDOW でのプログラミングに慣れている方は、本節を読み飛ばしてもかまいません。

なお、巻末の APPENDIX A 章には SX31KIT の技術的情報をまとめておきましたので、必要に応じて参照してください[1]。

1)SX31KIT のインストール方法は第 2 章「インストール」(p.33) を参照してください。

SX31KIT の構成

SX31KIT は、インクルードヘッダとライブラリファイルから構成されています。

インクルードヘッダ

SX31KIT に付属しているインクルードヘッダを Table 3-1に示します[2]。

2)このほかにも、SXCALL.H, SXCALL.EQU, SXCALL.MAC がありますが、これらはアセンブラ用のマクロファイルです。コラム「アセンブラマクロファイル」(p.157) で説明してあります。

Table 3-1 ● SX31KIT 付属のインクルードヘッダ

ファイル名	内　容
COLOR.H	カラーマン関数の定義ファイル
DIALOG.H	ダイアログマン関数の定義ファイル
MENU.H	メニューマン関数の定義ファイル
PRINT.H	プリントマン関数の定義ファイル
SXDEF2.H	SX ライブラリ関数用の共通シンボル定義ファイル
SXGRAPH.H	グラフマン系の関数の定義ファイル
TASK.H	タスクマン関数の定義ファイル
TEXT.H	テキストマン関数の定義ファイル
VIDEO.H	ビデオマン関数の定義ファイル
WINDOW.H	ウィンドウマン系の関数の定義ファイル

これらのうち COLOR.H と VIDEO.H については SX-WINDOW ver.3.1 の SX コールを使うために新たに導入されたもので、Workroom にはありません。その他のインクルードヘッダは、Workroom 付属のものの上位互換となっていて、完全に置き換えることが可能です。

また、ここに挙げられていない CONSOLE.H, CONTROL.H, EVENT.H, RESOURCE.H, SXMEMORY.H については、Workroom のものをそのまま利用します。

SX31KIT では、Workroom と同様、マネージャごとにインクルードヘッ

ダが分割されています。アプリケーションのソースコードからは、使用するマネージャのインクルードヘッダだけを読み込みます。

たとえば、アプリケーションでメニューマンの関数を使用するときは、ソースコードから、

```
#inlcude <MENU.H>
```

としてインクルードヘッダを読み込みます。

ライブラリファイル

SX31KIT に付属するライブラリファイルを、Table 3-2に示します。

Table 3-2 ● SX31KIT 付属のライブラリファイル

ファイル名	内　容
SX3LIB.A	AR.X でアーカイブしたライブラリファイル

SX3LIB.A には、SX-WINDOW ver.3.0/3.1 で追加された SX コールのみが含まれています。C 言語のスタートアップルーチンや、SX-WINDOW ver.2.0 以前の SX コールについては Workroom 付属の SXLIB.L を使用しなければなりません。

たとえば、foo.c というファイル名のプログラムをコンパイルする場合、コンパイラが XC ならば、

```
cc foo.c __mainr.o sx3lib.a sxlib.l
```

とします[3]。コンパイラが gcc ならば、

```
gcc foo.c __mainr.o -lsx3 -lsx
```

のようにします。

SX31KIT では、ライブラリファイルは XC ver.1 で使用されていた AR 形式になっていますが、XC ver.2 をお持ちの方は、LIB.X を用いて次のように変換作業を行うと、XC ver.2 相当の LIB 形式に変換することができます。

```
lib sx3lib.l sx3lib.a
```

なお、この作業は添付 FD のインストーラが行いますので、ユーザが直接行う必要はありません。

3)インストーラの質問への答えによって、libsx3.a, libsx.l になる場合もあります。

SX31KIT を使ったサンプルプログラム

では、SX31KIT を使うと、どんなプログラムが開発できるのでしょうか？

ここでは、階層メニュー[4)]を扱ったサンプルプログラムを取り上げます。

4) コラム「階層メニュー」を参照。

まず、SX31KIT が正しくインストールされているかどうかの確認の意味もかねて、サンプルプログラムをコンパイルしてみましょう。コンパイル作業は SX-WINDOW 上ではなく、コマンドライン上で行います。

サンプルプログラムをコンパイルするための作業ディレクトリを用意します。ここでは A:¥work を作業ディレクトリとします。

```
A:¥> mkdir work
A:¥> cd work
```

次に本書添付 FD の ¥sample¥sample ディレクトリから、skeleton.c, sample.c, sample.h という 3 つのファイルを、用意した作業ディレクトリにコピーします。ここでは B ドライブに添付 FD があるものとします。

⊙コラム：階層メニュー

SX-WINDOW ver.3.0 になって、階層メニューが正式にシステムでサポートされました。階層メニューとは、メニューアイテムとして別のメニュー（子メニュー）があり、その別メニューのアイテムとしてさらに別のアイテムがある、というようにメニューが階層的に構成されているもののことです。この説明でよくわからない場合は添付 FD 内の ¥SAMPLE サンプルディレクトリのサンプルプログラムを実行してみてください。

階層メニューの使い方を簡単に説明します。階層メニューのレコードの構成自体は、従来のメニューのレコードとまったく同じです。異なるのは、あるメニューアイテムに子メニューをつける場合、そのメニューアイテムのショートカットキーコードにエスケープ (0x1b) を、チェックマークに子メニューのリソース 'MENU' の ID を指定することです。つまり、子メニューはあらかじめリソースファイルに登録されていなければならず、しかも、そのリソースファイルがオープンされていなければなりません。メニューの表示や選択をする場合にも、今までのように、MNSelect を使用します。階層メニューのアイテムが選ばれた場合、返り値の上位ワードにその階層メニューのリソース 'MENU' の ID が、下位ワードにアイテム番号が格納されます。

```
A:¥work> copy b:¥sample¥sample¥skeleton.c a:
A:¥work> copy b:¥sample¥sample¥sample.c a:
A:¥work> copy b:¥sample¥sample¥sample.h a:
```

同時に、あなたが使用するコンパイラにあわせたバッチファイルもコピーします。コンパイラに gcc を使用するならば MAKEGCC.BAT を、XC を使用するならば MAKEXC.BAT を選択します。ここでは、コンパイラとして gcc を利用するものとします。

```
A:¥work> copy b:¥sample¥sample¥MAKEGCC.BAT a:
```

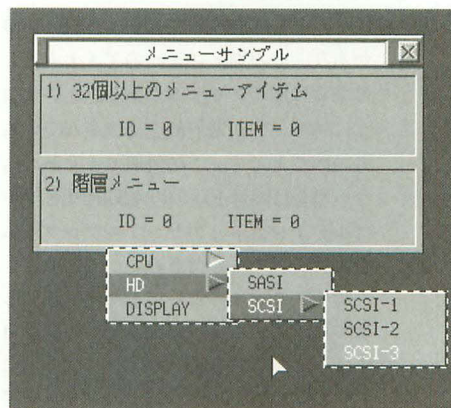
3つのファイルをコピーできたら、いよいよコンパイルです。

コマンドラインから MAKEGCC.BAT を実行します。もしあなたが XC を使っているなら、MAKEXC.BAT を実行してください。

```
A:¥work> MAKEGCC.BAT
```

SX31KIT およびコンパイラが正しくインストールされていれば、作業用ディレクトリに sample.x という実行ファイルができているはずです。

最後に、SX-WINDOW を起動してできあがった実行ファイルが正しく動作するかどうかを確認します。なお、sample.x の実行には sample.lb が必要ですので、先ほどと同様にして添付 FD からコピーしてください。

Fig. 3-1 ● sample.x の実行画面

アプリケーションの基本構造とスケルトン

ここでは、サンプルプログラムを通して SX-WINDOW アプリケーションの基本構造を具体的に説明します。また、SX31KIT の利用のしかたについても、必要に応じてふれていきます。

まず、SX-WINDOW アプリケーションの基本構造を簡単に復習します。「1.1 SX-WINDOW 概説」(p.4) でも述べたように、SX-WINDOW のプログラムはイベントを中心に動作しています。そのため、ほとんどのプログラムは Fig. 3-2のような構造を持ち、以下に示すような流れで処理を行います。

○前処理

変数の初期化

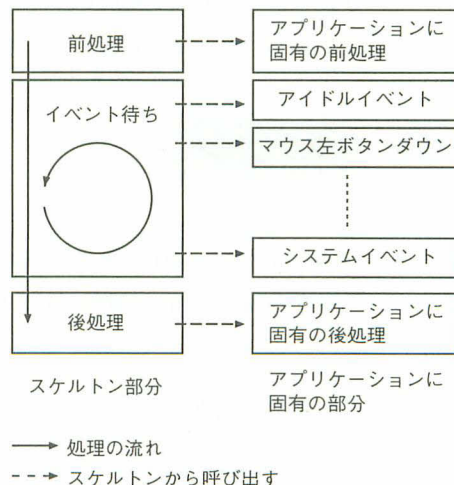
リソースファイルのオープン、ウィンドウの作成など。

○イベントごとの処理

発生したイベントに応じた処理を行う。

○後処理

リソースのクローズ、ウィンドウの破棄など。

Fig. 3-2 ● アプリケーションの基本構造

具体的に、どのような前処理を行うか (どのようなウィンドウをいくつ開くか、ボタンやスクロールバーなどのコントロールアイテムも作成

するのか)、個々のイベントに応じてどのような処理を行うのか (ボタンが押されたら音を出すとか)、どのような後処理を行うのか (アプリケーションの処理結果を保存するとか) については、アプリケーションによって違います。

しかし、どの SX−WINDOW プログラムも、基本的に Fig. 3-2のような構造を持っていることに変わりはありません。この構造は、いわば SX−WINDOW プログラムの**「骨格」**といえます。

この基本的な骨格部分は、どんなプログラムにもある程度共通しますから、新しいプログラムを作成するたびに最初から書くのはあまり能率がよいとはいえません。そこで、多くのプログラムに共通する骨格部分と、個々のプログラム固有の処理を行う部分とを分離します。この骨格部分を「スケルトン」[5)]と呼びます。

スケルトンには、リソースのオープンや、イベントに応じてアプリケーション固有の処理を呼び出す[6)]などの、アプリケーションに依存しない処理を記述しておきます。このスケルトンの部分は、一度作成すれば何度でも再利用が可能です。

スケルトンに含めることができないアプリケーション固有の処理 (ウィンドウの作成など) は、スケルトンとは別に記述します。新しいアプリケーションを作成するときは、この固有の処理部分のみを新たに作成すればいいわけです。

先の階層メニューを扱ったサンプルプログラムの場合、`skeleton.c` がスケルトンに、`sample.c` がアプリケーション固有の部分に相当します。それぞれを少し詳しく見ていくことにしましょう。

5)“skeleton” とは、英語で骨格を意味します。

6)「呼び出すだけ」です。具体的にどのような処理が行われるかについてはスケルトンは関知しません。

スケルトン

スケルトン `skeleton.c` を詳しく見ていきます。

スケルトンで行う処理には以下のようなものがあります。

○前処理

・変数の初期化

・SX−WINDOW のバージョンチェック

・リソースファイルのオープン

・コマンドラインの解析

・アプリケーション固有の前処理を呼び出す

○イベントごとの処理

・イベント待ちをし、イベントに応じたアプリケーション固有の処理を呼び出す。

○後処理

・アプリケーション固有の後処理を呼び出す

・リソースファイルをクローズする

次に、各項目ごとの処理を詳しく見ていきましょう。

インクルードヘッダの読み込み

まず、skeleton.c で利用するマネージャに応じて、SX31KIT のインクルードヘッダを読み込む必要があります。skeleton.c でインクルードヘッダを読み込む部分を List 3-1に示します。

List 3-1 ● skeleton.c: インクルードファイルの読み込み

```
15:  #include <stddef.h>      ┐
16:  #ifndef __LIBC__         │
17:  #include <class.h>       │(1)
18:  #endif                   │
19:  #include <SXDEF2.H>      ┘
20:
21:  #include <stdio.h>       ┐
22:  #include <stdlib.h>      │(2)
23:  #include <string.h>      ┘
24:
25:  #include <EVENT.H>       ┐
26:  #include <MENU.H>        │
27:  #include <SXGRAPH.H>     │(3)
28:  #include <SXMEMORY.H>    │
29:  #include <TASK.H>        │
30:  #include <WINDOW.H>      ┘
```

インクルードヘッダの読み込みは、大きく以下の3段階に分けられます。

(1) 互換性の維持 (15〜19 行目)

Workroom 以前の古いインクルードヘッダ[7]や、LIBSXC との互換性を維持するための、いわばおまじないです。

互換性を考える必要がない場合でも、とりあえず、このように記

7)ここでは SXDEF.H, SXLIB.Hのことを指す

述してください。

(2) 標準ライブラリ用のインクルードヘッダの読み込み (21〜23 行目)

C 言語の標準ライブラリを利用するためのインクルードヘッダを読み込みます。当然、この部分は使用する標準ライブラリによって、読み込むインクルードヘッダが変わります。

(3) SX31KIT のインクルードヘッダの読み込み (25〜30 行目)

アプリケーションで利用するマネージャに応じた SX31KIT のインクルードヘッダを読み込みます。この部分も利用するマネージャによって変わります。読み込む順番について特に制限はないので、必要なものを列挙するだけでかまいません。

● 変数の初期化

「1.1　SX-WINDOW 概説」(p.4) で述べたように、メモリ使用効率の面から実行プログラムは極力 OBJR 型にするのが理想です。XC の場合、C 言語を使って OBJR 型の実行プログラムを作成する場合、アプリケーションで使用する、タスクごとに確保する変数を 1 つの構造体としてまとめてしまう必要があります。

List 3-2 ● sample.h：タスクごとの変数を構造体にまとめる

```
21:  typedef struct {
22:    int quit;
23:    int EventMask;
24:    TsEvent EventRec;
25:    Window* WinPtr;
26:    Rect WinRect;
27:    int ActiveFlag;
28:    int menuSel1;
29:    int menuSel2;
30:  } glVal;
```

さらに main 関数のところで、この構造体の変数をローカル変数として確保します。ローカル変数として確保することで、タスクごとに変数を独立させることができます。

List 3-3 ● skeleton：タスクごとの変数をローカル変数として確保する

```
65:  main (void)
66:  {
     (中略)
76:    glVal gv;
```

しかし、ローカル変数は他の関数から参照することができません。それでは困るので、各関数の引数に、この構造体へのポインタを追加します。各関数からはこのポインタを経由してローカル変数にアクセスします。

List 3-4 ● sample.h：ローカル変数へのポインタを引数に加える

```
36:  void appTini (int, glVal*);
37:  int appInit (glVal*, Rect*);
38:  void IdleEvent (glVal*);
39:  void MsLDownEvent (glVal*);
40:  void MsRDownEvent (glVal*);
```

SX–WINDOW のバージョンチェック

SX–WINDOW はバージョンが上がるにつれ、さまざまな機能が SX コールの形で追加されてきました。

たとえば、マルチフォントテキストは ver.3.0 以前の SX–WINDOW では使用できないので、アプリケーションでマルチフォントテキストを使用する場合、必ずバージョンをチェックしなければなりません。

バージョンチェックはアプリケーションに固有の処理のように見えますが、スケルトン自体が使用している SX コールのなかにもバージョンに依存するものがあります。また、SX–WINDOW ver.3.0 以降と、それより前とでは機能に大きな差があります。ユーザのなかには ver.2.0 を愛用している方もいらっしゃるようですが、本書は「SX–WINDOW ver.3.1 開発キット」と銘打っていることからもわかるように、`skeleton.c` では ver.3.1 以降であることをチェックします。

List 3-5 ● skeleton.c:SX–WINDOW のバージョンチェック

```
82:        if (SXVer () < SXVER3)
83:        {
84:            DMError (D_CONFIRM,
                       "SX-WINDOW ver.3.1 以降を使用して下さい。");
```

```
85:         return -1;
86:  }
```

バージョンのチェックにはタスクマンのSXVer関数を使います。skeleton.cではSX-WINDOWのバージョンが古かった場合、ダイアログマンのDMError関数で警告を出して終了するようになっています。

リソースファイルのオープン

アプリケーションリソース[8]をオープンします。そして、リソースのファイル名を決定します。通常、アプリケーションの実行ファイル名の拡張子を“.LB”に変えたものをリソースのファイル名とします。たとえば、実行ファイル名が“キャンバス.X”なら、リソースファイル名は“キャンバス.LB”となります。

8)コラム「リソース」を参照。

List 3-6 ● skeleton.c:リソースファイル名の変更

```
92:  TSGetTdb (&tdb, TS_OWN);
93:  deletesuffix (tdb.name, resFname);
94:  strcat (resFname, ".LB");
```

skeleton.cでは、タスクマンのTSGetTdb関数を呼び出して実行ファイル名を取り出し、その拡張子を“.LB”に変更しています.

リソースのファイル名が決まったら、タスクマンのTSResOpen関数を呼び出して、リソースファイルをオープンします。

List 3-7 ● skeleton.c:リソースファイルのオープン

```
 99:  resHdl = TSResOpen (resFname);
100:  if (resHdl == NULL)
101:    {
102:      DMError (D_CONFIRM, "リソースがオープンできません。");
103:      return -1;
104:    }
```

skeleton.cではTSResOpen関数の返り値をチェックしてエラーが起きた場合にはメッセージを出して終了するようにしています。

⊙コラム：リソース

SX–WINDOW におけるリソースとは、簡単にいうと交換可能な部品 (データ) のようなものです。

たとえば、ウィンドウ (定義関数) がそうです。SX–WINDOW には数種類のそれぞれ異なった外見を持つウィンドウがあります。ウィンドウの外見を描画しているのはウィンドウ定義関数と呼ばれるリソースで、システムのリソースファイル (SYSTEM.LB) に含まれています。もしウィンドウの外見を変えたければ、このウィンドウ定義関数のリソースを別のものに交換します。このとき書き換えるのはリソースファイル (中のリソース) だけで、SX–WINDOW のシステム (FSX.X) 自体を書き換える必要はありません。

次にあなたがアプリケーションを作っている場合を考えます。ウィンドウを開く位置や大きさは、プログラムを作成していくうちに何度となく変わるものです。もしウィンドウを開く位置や大きさをプログラム中に埋め込んでしまったらどうなるでしょう？　ウィンドウの位置をちょっと変えるだけでもプログラムをコンパイルし直さなければなりません。これは非常に面倒です。このような場合、ウィンドウを開く位置や大きさといった情報をリソースとして、アプリケーションのリソースファイル中に置くようにします。このようにしておけばプログラムをコンパイルし直さなくても、リソースを変更するだけでウィンドウの位置を手軽に変えることができます。

このように変更される可能性のある部分をリソースとしてプログラムから切り離しておくと、プログラムの外見や振る舞いを簡単に変更できるようになります。

●コマンドラインの解析

SX–WINDOW のアプリケーションでは、コマンドラインからウィンドウを開く位置を指定できるようになっています。たとえば、“キャンバス.X” に対して、

```
キャンバス.X -w100,100,300,250
```

とコマンドラインで指定すると、画面の左上から (100, 100) の位置にウィンドウが 200 × 150 ドットの大きさで開きます。

この処理は SX–WINDOW がやってくれるわけではなく、アプリケーションが自前でやらなければなりません。そのため、このような処理を支援する SX コール TSTakeParam が用意されています。

List 3-8 ● skeleton.c：コマンドラインの解析

```
117:  ret = TSTakeParam (tdb.command, &winRect, NULL, 0, NULL,
      NULL);
```

```
118:  if ((ret & 1) == 0)
119:    {
120:      /* ウィンドウ位置指定がない */
121:      winRect.l.l_t = TSGetWindowPos ();
122:      winRect.d.right = winRect.d.left + WIN_XSIZE;
123:      winRect.d.bottom = winRect.d.top + WIN_YSIZE;
124:    }
```

タスクマンの TSTakeParam 関数にコマンドラインを引数として渡すと、コマンドライン中に“-*w*”オプションによるウィンドウ位置の指定がないかどうかをチェックし、その結果をレクタングルとして返します。

skeleton.c ではさらに、“-*w*”オプションによるウィンドウ位置の指定がなかった場合、タスクマンの TSGetWindowPos 関数を利用して適当なウィンドウ位置を取得しています。

アプリケーションに固有の前処理を呼び出す

前処理の最後に、アプリケーションに固有の前処理を呼び出します。

すでに述べたように、アプリケーション固有の部分で具体的に何が行われるかについては、スケルトンの側はいっさい関知する必要がありません。そのため、基本的にはアプリケーション固有の前処理を行う関数を呼び出すのみです。

List 3-9 ● skeleton.c:アプリケーション固有の前処理を呼び出す

```
130:  if (!appInit (&gv, &winRect))
131:    {
132:      DMError (D_RED | D_CONFIRM, "アプリケーションの初期化に失敗し
          ました。");
133:      TSResRemove ();
134:      appTini (-1, &gv);
135:      return -1;
136:    }
```

skeleton.c では、アプリケーション固有の前処理でエラーが起きた場合を想定し、エラーが起きた場合にはメッセージを出して終了するようにしています。

イベントに応じた処理

SX−WINDOW アプリケーションの中心であるイベントに応じた処理の部分です。

ここでは、タスクマンの TSEventAvail 関数を呼び出してイベントを待ち、さらにイベントに応じたアプリケーションに固有の処理を呼び出します。

List 3-10 ● skeleton.c:イベント待ちループ

```
144:  while (!gv.quit)
145:  {
146:      TSEventAvail (gv.EventMask, &gv.EventRec);
147:      switch (gv.EventRec.ts.what)
148:      {
149:        case E_IDLE:      IdleEvent (&gv);     break;
150:        case E_MSLDOWN:  MsLDownEvent (&gv); break;
151:        case E_MSRDOWN:  MsRDownEvent (&gv); break;
152:        case E_KEYDOWN:  KeyDownEvent (&gv); break;
153:        case E_UPDATE:   UpdateEvent (&gv);  break;
154:        case E_ACTIVATE: ActiveEvent (&gv);  break;
155:        case E_SYSTEM1:
156:        case E_SYSTEM2:  SystemEvent (&gv);  break;
157:        default:                             break;
158:      }
159:  }
```

イベントに応じたアプリケーションに固有の処理部分は、実際には sample.c のほうに記述されており、skeleton.c ではプロトタイプの宣言のみ行っています。

ここでもアプリケーション固有の部分で具合的に何が行われるかについては、スケルトンの側はいっさい関知していないことに注意してください。

アプリケーションに固有の後処理を呼び出す

アプリケーションに固有の前処理を呼び出したときと同様に、アプリケーションに固有の後処理を呼び出します。

List 3-11 ● skeleton.c:アプリケーション固有の後処理を呼び出す

```
165:  appTini (0, &gv);
```

後処理 appTini 関数[9]でエラーが発生することも考えられますが、結局、アプリケーションが終了することには変わりはないので、skeleton.c ではエラーチェックをしていません。

9)"Tini" は "Init" を逆に並べたものです。前処理が Init なので、後処理は Tini というわけです。

● リソースファイルをクローズする

アプリケーションリソースをクローズします。これは、単純にタスクマンの TSResRemove 関数 を呼び出すだけです。

List 3-12 ● skeleton.c:リソースファイルをクローズする

```
170:  if (ResHdl)
171:        TSResRemove ();
```

アプリケーションに固有の部分

それでは、アプリケーションに固有の部分 sample.c を詳しく見ていきます。

アプリケーションに固有の部分で行う処理には以下のようなものが考えられます。

○前処理

　変数の初期化、メニュー、ウィンドウの作成などを行う。

○イベント処理

　個々のイベントに応じた処理を行う。

○後処理

　メニュー、ウィンドウの破棄などを行う。

ここに挙げた処理はサンプルプログラムで行っているもので、実際には個々のアプリケーションによって千差万別です。そのため、この後、各項目ごとの処理を見ていきますが、詳細にはふれません。みなさんがア

プリケーションを作成する場合には、自分の力でプログラムを組み立てていかなければなりません。

前処理

アプリケーションに固有の前処理は sample.c の appInit 関数で行っています。ここでは、まずアプリケーションに固有の変数の初期化を行います。

次に、アボート処理を行う関数を登録します。アボート処理関数とは、なんらかの理由で SX-WINDOW のシステムに異常が生じ、アプリケーションの実行を中断しなければならないときに呼び出される関数です。

アボート処理関数では、通常のアプリケーション終了処理と同等の処理を行えばよいことが多く、sample.c でもアプリケーション固有の後処理を行う appTini 関数をアボート処理関数として登録しています。

List 3-13 ● sample.c:アボート処理関数を登録する

```
166:  TSSetAbort (appTini, (long) & gv);
```

次にウィンドウを開きます。sample.c ではウィンドウに関して特別なことはしていないので、単純にウィンドウマンの WMOpen2 関数[10]を呼び出してウィンドウを開いています。

10) 従来の WMOpen と、新しい WMOpen2 と WHOpen とでは引数が若干変わっています。

List 3-14 ● sample.c:ウィンドウを開く

```
181:  gv->WinPtr = WMOpen2 (NULL, &gv->WinRect, winTitle, 0,
182:                        WI_STD2, 0, (Window*) -1, -1, TSGetID ());
183:  if (gv->WinPtr == NULL)
184:  {
185:      DMError (D_RED | D_CONFIRM, "ウィンドウが作成できません。");
186:      return 0;
187:  }
```

個々のイベントに応じた処理

アプリケーションに固有の部分では、アイドルイベントやマウスイベントなどの個々のイベントに応じた処理を記述します。これらの処理は、すでに述べたスケルトンから IdleEvent や MsRDownEvent などの関数

名で呼び出されます。対応する必要のないイベントについては、何もしない関数を用意しておきます。

ここでは、サンプルプログラムで特徴的なマウスライトダウンイベントについてのみ説明します。

マウスライトダウンイベントの処理は MsRDownEvent 関数で行われます。

まず、発生したイベントが本当に自分のウィンドウに対するものかどうかをチェックします。

List 3-15 ● sample.c:自分のウィンドウへのイベントかどうかを確かめる

```
251:  if (gv->EventRec.ev.whom.win ! == gv->WinPtr)
```

次に、イベントが発生したときのマウスポインタの座標から表示すべきメニューを決定します。サンプルプログラムでは、マウスの右ボタンを押す場所に応じて 2 種類のメニューのうちの 1 つが表示されます。

List 3-16 ● sample.c:右ボタンが押された場所を確かめる

```
 85:  pt = GMGlobalToLocal (gv->EventRec.ev.where.x_y);
 86:  if (GMPtInRect (&menuRect1, pt))
 87:  {
 88:      menuStr = menuStr1;
 89:      menuSel = &gv->menuSel1;
 90:  }
```

イベントの発生したマウス座標はイベントレコード中にあります。このマウス座標はグローバル座標系なので、グラフマンの GMGlobalToLocal 関数でローカル座標系に直す必要があります。

グラフマンの GMPtInRect 関数で、どちらのメニューを表示するかを決定します。

表示すべきメニューが決まったら、メニューレコードを作成します。これにはメニューマンの MNConvert2 関数を利用します。

List 3-17 ● sample.c:メニュー文字列からメニューレコードを作成

```
102:  menuHdl = MNConvert2 (NULL, menuStr, MI_PLN);
103:  if (menuHdl == NULL)
104:  {
105:      DMError (D_RED | D_CONFIRM, "メニューが作成できません。");
106:      return;
```

```
107: }
```

最後にメニューレコードに従ってメニューを表示、メニューアイテムの選択をします。メニューの表示、選択という一連の処理は、メニューマンの MNSelect 関数を利用します。

List 3-18 ● sample.c:メニューの表示とセレクト処理

```
110:  *menuSel = MNSelect (menuHdl, gv->EventRec.ev.where.x_y);
111:  MMHdlDispose ((Handle) menuHdl);
```

選択されたメニューアイテムの番号が、MNSelect 関数の返り値となります。

後処理

一般に後処理で行うことは、前処理で行ったことにそれぞれ対応します。sample.c でも同様で、アプリケーションに固有の後処理を行う appTini 関数では、appInit 関数でオープンしたウィンドウをクローズしています。

List 3-19 ● sample.c:ウィンドウの廃棄

```
142:  if (gv->WinPtr)
143:       WMDispose (gv->WinPtr);
```

むすび

ここで紹介したスケルトンは説明のための非常にプリミティブなものです。SX-WINDOW プログラミングに不慣れな人は、ここで紹介したスケルトンや他のサンプルプログラムを改造したり、少しずつ機能を付け足したりしていくのがよいでしょう。そうすることで SX-WINDOW に対する理解がより深まります。

もっと実用的なアプリケーションを作成する際は、Method-SX[11] などの高機能のスケルトンを利用し、アプリケーション固有の処理部分の作成に力を注ぐようにするとよいでしょう。

11) 仁泉大輔氏作の超高機能スケルトン。添付 CD-ROM に収録されています。

SX-WINDOW 3.1環境でのプログラミング例

本節では、SX-WINDOW ver.3.0 以降で拡張あるいは追加されたマネージャのうち、グラフィック画面に関するものについて解説します。

グラフィック画面に関する拡張には、

○ グラフィックウィンドウによる 65,536 色表示への対応

○ 静止画像データの伸張/圧縮のサポート

○ アニメーション動画 (CGA)[1] の再生/作成

1) CGA: Computer Graphic Animation

があります。

これらの拡張について詳細に説明するのは紙幅の都合上不可能なため、サンプルプログラムを例にした概説という形式をとりました。

本書の添付 FD には、各サンプルプログラムのソースファイルと実行ファイルが収められています。実際にサンプルプログラムを実行したり、ソースファイルを参照したりしながら、以下の説明を読むことをお勧めします。

グラフィックウィンドウ

SX-WINDOW ver.3.0 では、65,536 色の表示が可能なグラフィックウィンドウが用意されました。それまでグラフィック画像は 16 色までしか表示できず、65,536 色のグラフィック画像を表示するには減色処理をする必要がありました。

新しく用意されたグラフィックウィンドウは 768 × 512 ドットの画面モードで、512 × 512 ドットの範囲内で自由な大きさの 65,536 色グラフィック画像表示が可能です。

GRW.X

新しく用意されたグラフィックウィンドウは、実は GRW.X という 1 つ

の独立したアプリケーションによって実現されています。

Fig. 3-3 ● GRW.X は画面の中の画面？

GRW.X を起動すると、Fig. 3-3 のようなグラフィックウィンドウが現れます。よく見ると、ウィンドウの外枠が今までのものと違っていることがわかります。

ここで キャンバス.X を起動すると、もう 1 つの違いに気づきます。Workroom のウィンドウがグラフィックウィンドウの中に現れるのです。さらにキャンバス.X のウィンドウを移動してみると、グラフィックウィンドウによってクリッピングされていることがわかります。つまり、ウィンドウが階層[2]になっているのです。

2) コラム「階層ウィンドウ」(p.94) を参照。

さて、キャンバス.X のようにグラフィックウィンドウの中に子ウィンドウを開き、グラフィックを表示するにはどのようにすればよいのでしょうか。グラフィックの表示については、静止画像やアニメーション動画の解説で具体的にふれることにして、ここではグラフィックウィンドウの中に子ウィンドウを開く方法のみを説明します。

⦿コラム：階層ウィンドウ

階層ウィンドウを一言で説明すると、「ウィンドウの中にウィンドウがあって、その中にさらにウィンドウがあって…」ということです。

たとえばFig. 3-4では、ウィンドウAの中にウィンドウBがあって、そのウィンドウBの中にさらにウィンドウCがあります。また、ウィンドウCはウィンドウBによってクリッピングされ、右上の部分しか見えていません。

それに対してFig. 3-5は、階層ウィンドウではない例です。ウィンドウEはウィンドウDによってクリッピングされていますが、ウィンドウDの中にウィンドウEがあるわけではありません。

このようにウィンドウが階層的に配置されている場合、ウィンドウ間の上下関係が重要になります。この階層ウィンドウの上下関係は親子関係になぞらえられることが多く、「親ウィンドウ」、「子ウィンドウ」という呼び方をします。

Fig. 3-4の例では、Aが親ウィンドウで、Bが子ウィンドウとなります。同様に、Bが親ウィンドウで、Cが子ウィンドウです。親子関係は相対的なものですから、Bは親であると同時に子でもあるわけです。また、Aを親ウィンドウとしたとき、Cを孫ウィンドウと呼ぶこともあります。「親亀の背中に子亀を載せて～」というわけです。

Fig. 3-4 ● 階層ウィンドウの例

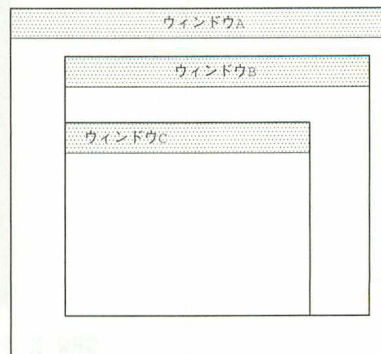
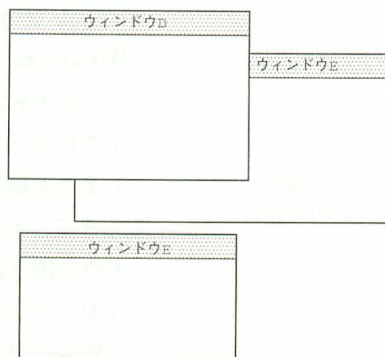



Fig. 3-5 ● 階層ウィンドウではない例

グラフィックウィンドウの開き方

先ほど述べたように、65,536色表示を司るGRW.Xは1つの独立したアプリケーションです。SXコールの形で実装されていないので、その利用には若干変則的な方法を用います。以下にその手順を示します。

①タスク間通信でGRW.Xのウィンドウポインタを取得する

これから開こうとしているグラフィックウィンドウは階層ウィンドウ

ですから、親ウィンドウへのポインタが必要となります。親ウィンドウはGRW.Xのウィンドウですから、どうにかしてGRW.Xのウィンドウへのポインタを知る必要があります。それにはタスク間通信を用います。

Fig. 3-6 ● GRW.X とのタスク間通信

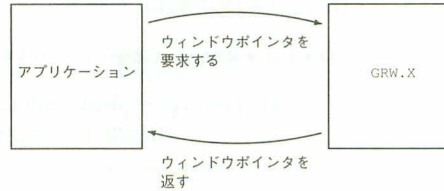



Fig. 3-6のように、GRW.Xのウィンドウの中にグラフィックウィンドウを開きたいアプリケーションの側からメッセージを送ります。

List 3-20 ● グラフィックウィンドウを開くアプリケーションからメッセージを送る

```
1:  TSPostEventTsk (WR_GROW, WR_WIND, REQRESPONSE, 0, 0);
```

GRW.Xが起動されるタイミングは不定ですから、通信相手のタスクIDを指定することは不可能です。そのため、TSPostEventTsk関数を用い、すべてのタスクに対してメッセージを送ります。REQRESPONSEは、返信要求のある通信を示すイベントです。ここではWR_GROW, WR_WINDをその引数とすることで、GRW.Xに対してウィンドウのポインタを返信するように要求しています。

②次にGRW.Xからの返信を待ちます。

返信を待つ手順は通常のタスク間通信の場合と同様、TSEventAvail関数でイベント待ちのループを作ります。アイドルイベントは、GRW.Xが起動していないことを意味します。この場合は、グラフィックウィンドウを開くことはできませんから、「GRW.Xが起動していない」というエラーメッセージを表示するダイアログを出して終了するか、あるいは自前でGRW.Xを起動します。

GRW.Xとの通信に成功すると、システムイベントのRESPONSEが回ってきます。この場合、イベントレコードのwhomを見て、相手がGRW.Xであることを確認したあと、イベントレコードのwhom2をGRW.Xのウィンドウへのポインタとして利用します。

③階層ウィンドウを開くには、新しいSXコールWHOpenを用います。

WHOpenは親ウィンドウへのポインタが引数に加わった以外、従来のWMOpenと同じです。親ウィンドウへのポインタには先ほど取得したGRW.Xのウィンドウへのポインタを指定します。また、ウィンドウ定義関数のIDにはWIN_CHILDを使用します。

List 3-21 ● WHOpenで階層ウィンドウを開く

```
 1:  WHOpen (gv->pWin, NULL, &gv->winRect, gv->winTitle, 0,
 2:          WIN_CHILD, W_SBOX, (Window *) -1, -1, TSGetID ());
```

WHOpenで作成したウィンドウに対しては、従来のWMOpenで作成したウィンドウと同様の扱いをすることができます。たとえば、ウィンドウ内部のアップデートの際はWMUpdate/WMUpdtOverを、ウィンドウの破棄にはWMDispose/WMCloseを使います。

List 3-22 ● GRW.Xからの返信を待つプログラム例

```
 1:  flag = 1;
 2:  while (flag)
 3:    {
 4:      TSEventAvail (EM_EVERY, &gv->myEvent);
 5:      switch (gv->myEvent.ts.what)
 6:        {
 7:        case E_IDLE:
 8:          /*
 9:           * E_IDLE イベントが回ってきたら GRW はいない。
10:           */
11:          flag = 0;
12:          break;
13:        case E_SYSTEM1:
14:        case E_SYSTEM2:
15:          switch (gv->myEvent.ts.what2)
16:            {
17:            case ENDTSK:
18:              EndProc (0, gv);
19:              break;
20:            case RESPONSE:
21:              /*
22:               * GRW から返事があった。
23:               */
24:              if (gv->myEvent.ts.whom == WR_GROW)
25:              {
26:                  /*
27:                   * GRW のウィンドウポインタを得る。
28:                   */
29:                  gv->pWin = (HWindow*) gv->myEvent.ts.whom2;
```

```
30:                   flag = 0;
31:                 }
32:               break;
33:             default:
34:               break;
35:             }
36:           break;
37:         default:
38:           break;
39:         }
40:     }
```

グラフィックウィンドウを使ったサンプルプログラム

グラフィックウィンドウの利用は、それだけでは単独のアプリケーションにはならず、あくまで他のアプリケーションの一部[3)]となるべき性格のものです。そこで、グラフィックウィンドウを使ったサンプルについては、ここでは紹介しません。そのかわり、このあとで登場する静止画像の再生やアニメーション動画の再生のサンプルプログラム中でグラフィックウィンドウを利用していますので、そちらを参照するようにしてください。

3) キャンバス.X や CG ビジョン.X がよい例です。

静止画像データの伸張/圧縮

SX-WINDOW ver.2.0 まではグラフィック画面が 16 色モードのみだったため、画像データのフォーマットとしては PIX 形式のみがサポートされていました。そのため、PIC 形式などの X680x0 では比較的メジャーなフォーマットの画像を扱うには、自前で画像ファイルの伸張ルーチンを書いたり、65,536 色から 16 色への減色を行う必要がありました。

SX-WINDOW ver.3.0 では、従来の PIX 形式に加え、TIFF, JPEG, PIC といった、 X680x0 でよく利用される画像フォーマットがサポートされました。これらのフォーマットに対する伸張/圧縮ルーチンはそれぞれモジュールとして独立しており、高機能のモジュールへの差し替えや、新しいモジュールの追加が簡単にできるように配慮されています[4)]。あとで述べる動画像においても、1 枚 1 枚の画像は静止画像[5)] として扱われます。

4) モジュールについてはコラム「IVM と画像モジュール」(p.98) を参照。

5) アニメーションと同じ要領で動画を実現しています。

⊙コラム：IVM と画像モジュール

ビデオマネージャ(IVM.X) は、各画像形式の伸張/圧縮プログラム (モジュール) を IVM.LB 中にコードリソースとして管理しています。画像形式ごとに VMIF と VMEC という 2 種類のリソースタイプがあります。

VMIF は、各画像形式のフォーマットチェックや画像の大きさ、色数等の情報取得を行います。一方、 VMEC は実際の伸張/圧縮を行います。

画像モジュールはリソースですので、ユーザが自由に交換・追加を行うことができます。本書では画像モジュールの詳細や作成方法はいっさい説明しません。そのかわり、すでにフリーソフトとして配布されているいくつかの画像モジュールを添付 CD-ROM に収録してあります。そのうちのいくつかの画像モジュールについてはソースコードもいっしょに収録してありますので、そちらを参照するようにしてください。

これら静止画像および動画像に関する処理は、IVM.X というプログラムによって実現されています。IVM.X は「ビデオマネージャ」と呼ばれ、フォントマネージャ IFM.X と同様、SX シェルから起動して常駐させることで、SX-WINDOW の機能を拡張します。

ここでは画像の伸張/圧縮の手順を示すとともに、「グラフィックウィンドウ」の項ではふれなかったグラフィックの表示方法を説明します。なお、以後、特に必要のないかぎり、静止画像を単に「画像」と表記します。

画像の伸張

画像の伸張の手順を以下に示します。

①画像ファイルをメモリ上に読み込む

最初に、画像ファイルをメモリ上に読み込みます。読み込む先はポインタでもハンドルでもかまいません。ポインタの場合、あとで疑似ポインタとして扱います。ここでは画像ファイルをハンドルに読む込むことにします。

まず、画像ファイルを開き、ファイルサイズを調べます。

List 3-23 ● 画像ファイルを開き、ファイルサイズを調べる

```
1:  fh = TSOpen ("SAMPLE.PIC", 0);
2:  fsize = SEEK (fh, 0, 2);
3:  SEEK (fh, 0, 0);
```

調べた画像ファイルの大きさのハンドルを確保します。

```
buf = MMChHdlNew (fsize);
```

確保したハンドルをロックし、実際にファイルを読み込みます。

```
MMHdlLock (buf);
READ (fh, *buf, fsize);
```

最後にファイルを閉じます。

```
TSClose (fh);
```

ここでは説明のため、細かいエラーチェックは省略しています。

②画像ファイルの情報を得る

次に①で読み込んだ画像ファイルに関する情報を得ます。

画像ファイルの情報は構造体 VideoInfo によって表され、画像の形式、サイズ、色数、パレットなどを含んでいます。

構造体 VideoInfo の各メンバとその内容を以下に示します。

Table 3-3 ● VideoInfo の各メンバと内容

型	メンバ名	内容
long	rscID	モジュールのリソース ID
char	idName[32];	モジュール名 (ASCIIZ 型)
char	idData[128];	モジュールに関する情報
char	expFName[4];	標準の拡張子
short	forPhoto;	VMGetInfo が可能かどうか
short	forMovie;	VMRegistSample が可能かどうか
short	withHeader;	画像形式を識別するヘッダがついているか否か
short	fixedBounds;	画像のレクタングルが固定かどうか
これより上は、必ず格納する領域		
short	bitsType;	最適なスクリーンタイプ
short	useBits;	色を表現するのに必要なビット数、フルカラーは−1
short	withPalette;	パレットをもつかどうか
Rect	baseRect;	想定されているベース画面のレクタングル
Rect	picRect;	実際の画像のレクタングル
long	expTime;	伸張に必要な時間
char	reserved[38];	システム予約
char	user[16];	モジュールに固有の領域

実際に①で読み込んだ画像ファイルの情報を得るには、読み込んだメモリへのハンドルと、画像ファイルのファイルネームへのポインタを引数として、VMGetInfo 関数を呼び出します。

List 3-24 ● VMGetInfo 関数の呼び出し

```
1:  VideoInfo info;
2:  moduleID = VMGetInfo (-1, buf, filename, &info, 0);
3:  if (moduleID < 0)
4:    {
5:      TSErrDialogN (D_CONFIRM,
6:                    "サポートされていないデータ形式です。");
7:    }
```

画像ファイルの情報取得に成功すると、VMGetInfo 関数は返り値として 1 以上の値を返してきます。この値を「モジュール ID」といい、画像の形式を表します。モジュール ID の値は、このあとの伸張で使います。

画像ファイルの情報取得に失敗すると、VMGetInfo 関数は負の値を返してきます。この場合、IVM でサポートしていない画像形式か、あるいは画像ファイルが壊れていることが考えられます。Table 3-4に SX-WINDOW ver.3.1 で正式にサポートされているモジュール ID と画像形式およびその標準拡張子の対応を示します。

Table 3-4 ● モジュール ID と画像形式・拡張子の対応

モジュール ID	画像形式	拡張子
1	TX16 16 色	PIX
2	EasyPaint 16 色	PNT
3	PAT4 パターンエディタ	PT4
4	ベタ (256×256 ドット) 65,536 色	GL0
5	ベタ (512×512 ドット) 65,536 色	GL3
6	ベタ (任意サイズ) 65,536 色	GLM
7	Runlength Pixel Chain	RPC
8	JPEG caseline DCT	JPG
9	TIFF モトローラ	TIF
10	TIFF インテル	TIF
11	APIC 16 ～ 65,536 色	PIC
12	画像ファイル (DōGA)	PIC
13	1Bit-per-pixel gray scale	OBM

③画像を伸張する

②で得られた情報をもとに VMExpand 関数を呼び出して画像を伸張します。

画像を伸張する先は Bits です。Bits には G_GRP タイプ[6]のものを自前で確保するか、NULL を指定して自動的に確保してもらうか、どちらかの方法をとります。いずれにせよ、G_GRP タイプの Bits に画像が伸張されることになります。

ここでは NULL を指定し、自動的に確保することにします。

6)グラフィックタイプの Bits。1 ドットにつき 2 バイトなので 65,536 色まで表現可能。

List 3-25 ● VMExpand 関数による画像の伸張

```
1:  Bits** img;
2:  img = VMExpand (NULL, buf, rscID, &info.picRect,
3:                  &info.picRect, 0);
4:  if (img == NULL)
5:    {
6:      TSErrDialogN (D_CONFIRM, "データの伸張に失敗しました。");
7:    }
```

VMExpand 関数の返り値として NULL が返ってきた場合、画像の伸張中にエラーが発生したことを表します。

画像の表示

次に、伸張した画像を 65,536 色のグラフィックウィンドウに表示します。グラフィックウィンドウを開く手順は「グラフィックウィンドウ」の項を参照してください。ここでは、すでにグラフィックウィンドウが開いているものとします。

まず、ウィンドウの内部のテキスト画面を透明色で塗りつぶします。これは、テキスト画面がグラフィック画面より表示の優先順位が高いため、透明色で塗りつぶしておかないとグラフィック画面が見えないからです。

List 3-26 ● テキスト画面の塗りつぶし

```
1:  GMAPage (15);
2:  GMForeColor (G_THRU);
3:  GMPenMode (G_PSET);
4:  GMFillRect (&gv->winPtr->graph.rect);
```

次にウィンドウの Bitmap をグラフィックタイプに差し替えます。テキスト画面ではなく、グラフィック画面に描画したいのですから、必ず差し替えます。

List 3-27 ● グラフィック画面への差し替え

```
1:  Bitmap* txtBmap;
2:  txtBmap = GMMakeGrpBitmap ();
```

65,536 色用のパレットを用意します。

List 3-28 ● 65,536 色パレットの準備

```
1: Palet** plt;
2: plt = CLRefer (PALT_G65536);
```

先ほど画像を伸張しておいた Bits をロックし、実際に描画をします。描画には GMCopy2 関数を用います。

List 3-29 ● 画像の描画

```
1: GMLockBits (img);
2: srcBmap = &(*img)->bmap;
3: dstBmap = GMGetBitmap ();
4: GMCopy2 (moduleID, srcBmap, dstBmap,
5:          &srcBmap->rect, &gv->winPtr->graph.rect,
6:          plt, plt, 0, NULL);
```

最後にウィンドウの Bitmap を戻したり、パレットを破棄するなどの後始末をします。

List 3-30 ● 後処理

```
1: MMPtrDispose ((Pointer) GMExgBitmap (txtBmap));
2: CLDisposePalet (plt);
3: GMUnlockBits (img);
```

描画の手順はやや複雑ですが、ほとんどの場合に同様の手順を踏む、いわば、定型パターンです。一度理解してしまえば、応用は難しくないでしょう。

画像の圧縮

画像の圧縮は、伸張の場合と逆の手順を踏みます。画像の描かれた Bits を VMCompress 関数に渡し、圧縮された画像ファイルを得ます。圧縮の際、どの画像形式で圧縮するかを指定する必要があります。これには伸張の説明でも出てきたモジュール ID を用います。

圧縮形式を明示的に指定するかわりに、システムのデフォルトの圧縮形式を指定することもできます。デフォルトの圧縮形式は IVM.X を 2 回以上起動すると現れるダイアログで設定します。

次に、画像の圧縮の具体的な手順を示します。

画像の描かれている Bits は伸張の場合と同様、G_GRP タイプでなければなりません。圧縮されたデータを格納するハンドルを指定することもできます。一般に圧縮されたあとのデータサイズは不定なので、どの程度の大きさのハンドルを確保するか悩むところですが、ハンドルの大きさが不十分な場合、自動的に拡張されるので、適当に数バイトの大きさのものを確保しておけばよいでしょう。また、ハンドルに NULL を指定すると自動的にハンドルが確保されます。

List 3-31 ● 画像の圧縮

```
1:  Bits** img; /* 画像の描かれている Bits */
2:  Handle hdl; /* 圧縮された画像データを格納するハンドル */
3:  hdl = VMCompress (img, NULL, -1, NULL, NULL, 0);
```

この例ではモジュール ID として −1 を指定し、デフォルトの圧縮形式で画像を圧縮しています。画像の圧縮に失敗すると、VMCompress 関数の返り値は NULL になります。

静止画像の伸張/圧縮のサンプルプログラム

静止画像の伸張を扱ったサンプルプログラム VMSMPL1.X を紹介します。

VMSMPL1.X はビデオマネージャが対応している形式の画像ファイルを伸張し、グラフィックウィンドウに表示します。

Fig. 3-7 ● VMSMPL1.X の画面

VMSMPL1.X はビデオマネージャとグラフィックウィンドウを利用するので、あらかじめビデオマネージャと GRW.X を起動しておきます。

VMSMPL1.X を起動するとウィンドウが開くので、ビデオマネージャで対応している形式の画像ファイルをドラッグし、ウィンドウにドロップします。VMSMPL1.X はドロップされた画像ファイルを伸張し、グラフィックウィンドウ中に 65,536 色で表示します。

静止画像の圧縮については、このあとの「CGA 作成のサンプルプログラム」(p.109) で取り上げていますので、そちらを参照してください。

アニメーション動画の再生・作成

アニメーション動画 (CGA) のサポートは、SX-WINDOW ver.3.0 の目玉機能の 1 つです。きちんと時間管理をしているので、たとえば 1 分の CGA は、どの X680x0 シリーズでも 1 分で再生されます。また、音声の同期もサポートされています。

CGA の再生や作成も、静止画像と同様、ビデオマネージャによって管理されており、SX コールを通して簡単に行うことができます。

CGA の再生

CGA を扱うには、まず、アニメーションポートをオープンします。

アニメーションポートとは、複数枚の画像データ、再生順序、音声トラック、トラック同期タイミング、画像メディアタイプ (メモリ上にあるか、ファイル上にあるか) といった CGA 再生に必要な一連のデータ群をまとめたものです[7]。

7) アニメーションポートの詳細は、残念ながら、公開されていません。

アニメーションポートにはメモリモードとファイルモードの 2 つの動作モードがあります。メモリモードを指定すると、管理情報や画像データを含めた CGA 全体がメモリ上に置かれます。そのため、再生や作成が高速に行えますが、すべてをメモリ上に置く必要があるため、空きメモリ以上の大きさの CGA を扱うことができません。

一方、ファイルモードを指定すると、管理情報のみをメモリ上に置き、画像データはファイル上に置きます。そのため、空きメモリの大きさ以上の巨大な CGA も扱うことができますが、再生の速度はファイルを格納しているデバイスに依存します[8]。

8) フロッピーディスク等の遅いデバイスを使うとコマ落ちが激しくなります。また、RAM ディスクを使うとファイルモードにする意味がありません。

それではまず、CGA の再生の手順を説明します。

既存の CGA ファイルをファイルモードで再生します。最初に CGA ファイル (SAMPLE.CGA) をオープンします。

List 3-32 ● CGA ファイルのオープン

```
1:  int fh;
```

```
2: fh = TSOpen ("SAMPLE.CGA", 0);
```

次に、アニメーションポートをファイルモードで作成します。CGA ファイルのファイルハンドルを引数にして VMCreateF 関数を呼び出します。アニメーションポートの作成に失敗すると NULL が返ります。

List 3-33 ● アニメーションポートの作成

```
1: Handle aport;
2: aport = VMCreateF (fh);
3: if (aport == NULL)
4:   {
5:     TSErrDialogN (D_CONFIRM,
6:                   "アニメーションポートの作成に失敗しました。");
7:   }
```

表示に備えて、アニメーションの縦横のサイズを調べます。VMGetWidthHeight 関数を呼び出すと、アニメーションのサイズを表すポイントが返ります。

List 3-34 ● アニメーションサイズを調べる

```
1: Point asize;
2: asize = VMGetWidthHeight ();
```

必要な情報を取り出したら、次にアニメーションポートをクローズします。CGA を再生するときには必ずアニメーションポートをクローズしなければなりません。

List 3-35 ● アニメーションポートのクローズ

```
1: VMClose (aport);
```

次に、CGA を表示するための準備をします。CGA の表示はグラフィックウィンドウを使用します。グラフィックウィンドウを開く手順は「グラフィックウィンドウ」(p.92) の項を参照してください。ここでは、すでにグラフィックウィンドウが開いているものとします。CGA 表示の準備は、静止画像の場合とまったく同じです。詳細は「画像の表示」(p.101) の項を参照してください。

List 3-36 ● CGA の表示の準備

```
1:  GMAPage (15);
2:  GMForeColor (G_THRU);
3:  GMPenMode (G_PSET);
4:  GMFillRect (&gv->WinPtr->graph.rect);
5:  txtBmap = GMMakeGrpBitmap ();
6:  palt = CLRefer (PALT_G65536);
7:  old = GMSetPalet (palt);
```

CGA の再生を開始します。再生には VMPlay 関数を呼び出します。

List 3-37 ● CGA の再生

```
1:  rect = gv->WinPtr->graph.rect;
2:  GMMoveRect (&rect, (LPoint) 0);
3:  VMPlay (gv->AnimHdl, &rect);
```

最後に後始末をします。

List 3-38 ● 後処理

```
1:  MMPtrDispose ((Pointer) GMExgBitmap (txtBmap));
2:  GMSetPalet (old);
3:  CLDisposePalet (palt);
```

以上で CGA の再生が始まります。これ以降はイベントに応じて処理を行います。実際にはアニメーションポートとイベントレコードを引数に VMEvent 関数を呼び出せば、あとはビデオマネージャが適切な処理を行います。

List 3-39 ● VMEvent 関数の呼び出し

```
1:  VMEvent (aport, &eventrec);
```

CGA の再生を終了するときはアニメーションポートを破棄し、CGA ファイルをクローズします。

List 3-40 ● CGA ファイルのクローズ

```
1:  VMDispose (aport);
2:  TSClose (fh);
```

CGA の再生は、再生を始めるまでが少し複雑ですが、一度再生を始めてしまえばあとは VMEvent 関数にほとんど任せることができます。

CGA の作成

最後に CGA の作成に挑戦してみましょう。

静止画像のところでも述べたように、CGA は複数枚の静止画像から構成されています。つまり、1枚1枚の静止画像を順次登録していくことで CGA ができあがります。なお、画像以外のメディア（FM 音源や ADPCM など）の登録方法は、残念ながら、わかっていません。ビデオマネージャ (IVM.X) には画像以外のメディアを登録する機能がないので、なんともしようがありません[9]。

CGA の作成手順を説明します。

まず、新しく作成する CGA のファイル名を決定し、その名前でファイルをオープンします。オープンしたファイルのファイルハンドルを引数に VMCreateF 関数を呼び出し、アニメーションポートをファイルモードで作成します。

List 3-41 ● CGA ファイルの作成

```
1: int fh;
2: Handle aport;
3: fh = TSCreate ("NEW.CGA", 2);
4: aport = VMCreateF (aport);
```

次にメディアを登録します。メディアには画像や音声などがありますが、現在、ビデオマネージャでは画像以外のメディアはサポートされていないので、メディア＝画像となります。メディアの登録は VMRegistSample 関数で行います。

登録する画像データは、メモリ上のハンドルに格納されている必要があります。そのため、ファイル上の画像データは一度メモリ上に読み込む必要があります[10]。また、1つのアニメーションポート内では、すべての画像は同一の形式でなければなりません。

List 3-42 ● 画像データの登録

```
1: Handle img; /* 画像の格納されているハンドル */
2: id = VMRegistSample (img, MMHdlSizeGet (img), -1, 0);
```

9) CPLK.X (DDZ 氏作製) というフリーソフトを使用すると、CGA ファイルに ADPCM データを追加することができます。添付 CD-ROM の ¥DEVELOP ディレクトリに収録されています。

10) 「画像の伸張」の項 (p.98) を参照。

メディアの登録に成功すると、メディア ID が返ります。メディア ID は 1 枚 1 枚の画像につけられた通し番号で、以降、このメディア ID で各画像を参照します。

画像データの登録が終わったら、次にタイムスケールの設定をします。タイムスケールとは、CGA 再生において再生時間を指定する際の単位を決定するパラメータです。タイムスケールが x の場合、時間の単位は $\frac{1}{x}$ 秒になります。タイムスケールの設定は VMSetTimeScale 関数で行います。

```
VMSetTimeScale (64);
```

この場合、時間の単位は $\frac{1}{64}$ 秒になります。タイムスケールは、このあとのモーションの登録に関係してきます。

次にモーションを登録します。モーションとは、登録された画像の再生順序および表示期間を指します。モーションの登録は VMInsertFrame 関数で行います。VMInsertFrame 関数では、メディア ID で示される画像を、どの位置に (何番目に)、どれだけの時間表示するかを指定します。表示する時間の単位は先ほど設定したタイムスケールで決まります。

```
VMInsertFrame (id, -1, 4);
```

この例では、id で示される画像を一番最後 (−1) に挿入しています。タイムスケールが 64 だとすると、単位時間は $\frac{1}{64}$ 秒ですから、この画像を $\frac{4}{64}$ 秒間、つまり $\frac{1}{16}$ 秒間表示することになります。なお、1 つの画像を何回指定してもかまいません。

次に CGA の総再生時間長を設定します。総再生時間は VMSetDuration 関数で設定します。単位はタイムスケールを基準にします。次の例では、全部で 10 枚の画像をそれぞれ表示時間 4 で表示するため、総表示時間を 40 に設定しています。

```
VMSetDuration (4 * 10);
```

最後に CGA をファイルに出力します。VMClose 関数を呼び出してアニメーションポートをクローズすると、自動的に CGA の内容がファイルに出力されます。忘れずにファイルをクローズするようにしてください。

List 3-43 ● CGA ファイルのクローズ

```
1: VMClose (aport);
2: TSClose (fh);
```

CGA 再生のサンプル

ここでは、CGA 再生を扱ったサンプルプログラム VMSMPL2.X を紹介します。

VMSMPL2.X は非常にシンプルな CGA 再生プログラムです。拡張子が ".CGA" のファイルを読み込み、グラフィックウィンドウ上で再生します。

Fig. 3-8 ● VMSMPL2.X の画面

VMSMPL2.X はビデオマネージャとグラフィックウィンドウを利用するので、あらかじめビデオマネージャと GRW.X を起動しておきます。

VMSMPL2.X を起動するとウィンドウが開くので、拡張子が ".CGA" の CGA ファイルをドラッグし、ウィンドウにドロップします。VMSMPL2.X はドロップされた CGA ファイルの再生をグラフィックウィンドウ上で始めます。ウィンドウ上で右ボタンをクリックするとポップアップメニューが現れ、ポップアップメニューからはウィンドウのサイズを元に戻したり、CGA ファイルをメモリに読み込んで高速に再生したりすることができます。

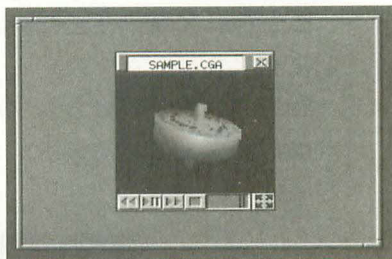
CGA 作成のサンプルプログラム

CGA 作成を扱ったサンプルプログラム VMSMPL3.X を紹介します。

VMSMPL3.X は、非常に簡単な CGA 作成のプログラム例です。CGA 作成の手順を示すことが主目的なので、これ自体にはほとんど応用性はありません。もっと実用的なもの (スクリプトを解釈するものや、インタラ

クティブに CGA を編集するものなど) を作る際の参考にしてください。

VMSMPL3.X と、添付 FD に含まれる拡張子が “.RPC” の画像ファイルを同じディレクトリに置いたのち、VMSMPL3.X を起動してください。ウィンドウは開かず、ダイアログのみ表示されます。無事 CGA が作成されると、同じディレクトリに SAMPLE.CGA というファイルができあがっているはずです。

Fig. 3-9 ● VMSMPL3.X で作成した CGA の再生画面

できあがった CGA ファイルは、先の VMSMPL2.X や、CG ビジョン.X を使って再生してみてください。

以上、SX-WINDOW ver.3.0 以降で拡張された機能のうち、グラフィック画面に関するものについて、その基本的な利用の手順を駆け足で説明してきました。

筆者の技量不足と紙幅の都合から細部まで十分に説明することができなかった箇所が多々あるかと思います。そういう部分に関しては、添付 FD に収録されているサンプルプログラムのソースリストを参照するようにしてください。

callno header

callno header は Workroom と互換性のあるインラインヘッダです。callno header を利用すれば、今までライブラリをリンクしていたプログラムの大半をコンパイルしなおすだけで、高速かつコンパクトな実行ファイルを作成することができます。

また、旧版のヘッダ[1]と互換性があるので、過去のプログラムを蘇らせることが可能です。

1)追補版の付録ディスクに収録されている SXDEF.H と SXLIB.H のことを指す。

インストール

callno header は、Table 3-5に示すファイルから構成されています。

Table 3-5 ● callno header を構成するヘッダファイル

ヘッダファイル名	内　容
Animation_i.h	アニメーションマン
Color_i.h	カラーマン
Control_i.h	コントロールマン
Dialog_i.h	ダイアログマン
Event_i.h	イベントマン
Exception_i.h	エクセプションマン
Font_i.h	フォントマン
Graph_i.h	グラフマン
Key_i.h	キーマン
Keyboard_i.h	キーボードマン
MText_i.h	テキストマン (マルチフォント)
Memory_i.h	メモリマン
Menu_i.h	メニューマン
Mouse_i.h	マウスマン
Print_i.h	プリントマン
Resource_i.h	リソースマン
Semaphore_i.h	セマフォマン
Subwindow_i.h	サブウィンドウマン
Task_i.h	タスクマン
Text_i.h	テキストマン
Video_i.h	ビデオマン
Window_i.h	ウィンドウマン
sxdecl.h	互換性を保つための各種マクロ定義

ヘッダファイル名	内　容
sxcall.equ	X680x0 gcc 用のインクルードファイル
include.dif	Workroom ヘッダに対するパッチ
sxlib.h	旧版との互換性を保つためのヘッダ

callno header が提供するのは関数宣言のみです。構造体定義などはいっさい行わないので、Workroom のヘッダ (あるいは旧版のヘッダ) が必要となります。

インストールを行う前には、念のため、自分の include ディレクトリのバックアップをとっておいたほうがよいでしょう。

Workroom 環境 (SX31KIT) の場合

ここでは 環境変数 include に設定されているディレクトリに Workroom のヘッダファイル (SXDEF2.H, SXGRAPH.H など) がすでにインストールされているものとします (以下、環境変数 include に設定されているヘッダファイルの存在するディレクトリを$include とする)。

まず、$include に sx という名前のサブディレクトリを作成します。

```
> cd $include
> mkdir sx
```

このサブディレクトリに callno header のすべてのヘッダファイル (*_i.h) をコピーします。

さらに、$include へ callno header の sxcall.equ をコピーします。すでに同名のファイルが存在し、これを消したくない場合には、callno header の sxcall.equ を別の名前に変更し、環境変数 SXEQU にそのフルパス名を指定してください。

最後に、Workroom のヘッダにパッチを当てます。これは、状況に応じて Workroom のヘッダが callno header を読み込むように設定するためです。 $include に付属の include.dif をコピーし、$include ディレクトリで patch.x を実行します。

```
> cd $include
> patch.x -p1 < include.dif
```

以上で、Workroom 環境 (SX31KIT) で使用する場合のインストールは終了です。

旧版ヘッダ (SXDEF.H) 環境の場合

ここでは、$include に旧版ヘッダがすでにインストールされているものとします。

まず、$include に sx という名前のサブディレクトリを作成します。

```
> cd $include
> mkdir sx
```

このサブディレクトリに callno header のすべてのヘッダファイル (*_i.h) をコピーします。また、旧版ヘッダの SXLIB.H もコピー (ないしは移動) します。

さらに、$include に callno header の sxlib.h をコピーします。

最後に、$include に callno header の sxcall.equ をコピーします。すでに同名のファイルが存在し、これを消したくない場合には、callno header の sxcall.equ を別の名前に変更し、環境変数 SXEQU にそのフルパス名を指定してください。

以上で、旧版ヘッダ (SXDEF.H) 環境で使用する場合のインストールは終了です。

両方の環境を併用する場合

callno header は Workroom 環境と旧版ヘッダ環境とで同時に使用することができます。この場合は、それぞれの環境に応じたインストールを行ってください。

利用方法

コンパイル方法

callno header の利用は、いたって簡単です。SXCALL 関数をインライン展開したい場合に、マクロ __SX_INLINE__ を定義してコンパイルするだけです。

a) ライブラリをリンクする場合

```
> gcc -SX -O sample.c -lsx
```

b) インライン展開をする場合

```
> gcc -SX -O sample.c -lsx -D__SX_INLINE__
```

b) のようにインライン展開指定を行ってコンパイルした場合には、ほとんどのSXコール呼び出しがインライン展開されます。この場合、'*-lsx*' としてライブラリをリンクする必要がないように思われますが、実際にはスタートアップをリンクするために必要です。また、CLEvent など、一部の関数はSXコールではなくライブラリですので、このような関数[2]についてもライブラリをリンクする必要があります。

2) 第2部のSXコールリファレンスを参照してください。

ライブラリとの互換性

ライブラリの一部(リソースマン関係)ではライブラリ変数 errno に値を返してくることがありますが、callno header ではそれを考慮していません。リソースマン関係の関数を呼んだあとで errno を参照しているようなプログラムは、callno header を利用してコンパイルすると誤動作する可能性があります。そのような場合は、callno header ではなく、ライブラリを用いる必要があることに注意してください。

callno header の欠点

callno header を利用してコンパイルすると、ほとんどの SXCALL 関数の名前がプリプロセッサの段階で _sx_???? の形式に変わります。そのため、コンパイラのエラーや警告においても _sx_???? としか表示されず、どの関数に対するエラーや警告なのかがわからなくなってしまいます。

対策としては、コール番号と関数名の対応を覚えておく、エラーや警告の発生したソースリスト中の行番号を頼りにするなどが考えられます。しかし、前者はすべての関数について覚えるのは不可能ですし、その都度確認するのも面倒です。筆者の場合は、後者の、行番号を利用してソースを参照し、どの関数でエラーが発生したのかを調べる手法をとっています[3]。

3) エラーメッセージ中の行番号を見て、その行を参照する手法は「タグジャンプ」と呼ばれています。

callno header の仕様

予約語 SXCALL と _SXCALLPtr

4)開発言語で、あらかじめ役割が決められている文字列。C 言語での *if* や *for* といった文字列のように、機能が決まっており、変数名としては使用できない文字列のこと。

X680x0 gcc の SX モードでは、予約語[4] SXCALL と _SXCALLPtr を用いることでライブラリを経由せずに SX-WINDOW のシステムコールを直接呼び出すことができます。

さて、この機能はどのように実装されているのでしょうか。

実際にこの機能を実現するという観点から考えてみると、クリアすべき点が 3 つ挙げられます。

(1) どのように A-line trap を発行するか。また、その番号をどのように指定するか。

(2) ワードサイズやバイトサイズの引数をどうやってスタックに積むか。

(3) A0 レジスタに返り値があるコールをどう扱うか。

以下、具体例を挙げながら、ひとつひとつ見ていきましょう。

1. A-line trap の発行

まずはじめに、用語の整理をしておきます。

以下では、SX-WINDOW のシステムコールを「SX コール」、予約語 SXCALL を用いて宣言された関数を「SXCALL 関数」と呼んでそれぞれを区別します。

さて、どうやって A-line trap を発行するかですが、ここでは説明を簡単にするため、引数も返り値も持たない SXCALL 関数を考えます (List 3-44)。

List 3-44 ● 引数、返り値のない SXCALL 関数

```
1:  SXCALL void foo (void); <--- SXCALL 関数の宣言

2:  void bar (void)
3:  {
4:    foo (); <--- SXCALL 関数の呼び出し
5:  }
```

SXCALL 関数は単なる宣言であって、通常の関数のようにその実体を

記述することはできません。当然、ポインタで扱うこともできません。コンパイルして得られるアセンブラソース上での SXCALL 関数の呼び出し部分は、ちょうどインライン展開されたようになります。

List 3-45 ● List3-44 のアセンブラソース

```
 1:  * NO_APP
 2:  RUNS_HUMAN_VERSION   equ     2
 3:      .cpu 68000
 4:      .include sxcall.equ <--- シンボルを記述してある
 5:  * X68 GCC Develop
 6:      .even
 7:      .text
 8:      .even
 9:      .globl _bar
10:  _bar:
11:      link a6,#0
12:      SXCALL foo <---- SXCALL の呼び出し
13:      unlk a6
14:      rts
15:      .even
16:      .end
```

“SXCALL foo” が、SXCALL の呼び出し部分で、関数名がアセンブラのシンボル扱いになっています。“SXCALL” はアセンブラのマクロで、

```
SXCALL  macro       callname
        .dc.w       callname
        endm
```

と sxcall.equ で定義されています。

あとはこのシンボルに A-line trap の値を定義すれば完成です。実際にそれを行っているのが sxcall.equ[5] なのです。

つまり、「C 言語のソース上ではなく、それとは別にアセンブラのインクルードファイル上で記述する必要がある」わけです。SX コールの名前とコール番号との対応は基本的に不変ですから、一度 sxcall.equ に記述してしまえば十分です。

5) これはデフォルトのファイル名です。別のファイル名に変更する場合は、環境変数 SXEQU で指定します。

2. 引数のサイズ

SXCALL では引数をスタックに積みます。この引数は、ワードサイズの場合とロングワードサイズの場合の 2 通りがあります。しかし、通常 C 言語の関数に対する引数[6] はすべてロングワードで積まれています。

6) 型が *char* や *short* であっても。

ライブラリでは、引数をスタックに積みなおすことでこの問題に対処していますが、これは実行時にかなりのオーバーヘッドとなります。そこで、SXCALL 関数では関数のプロトタイプ宣言を最大限に利用することで対処しています。

簡単な例を挙げて、その違いを見てみましょう (List 3-46)。

List 3-46 ● 引数のある SXCALL 関数

```
       void normal_func (int, short);
SXCALL void sxcall_func (int, short);

void foo (void)
{
  normal_func (1, 2);
  sxcall_func (1, 2);
}
```

List 3-47 ● List3-46 のアセンブラソース

```
* NO_APP
RUNS_HUMAN_VERSION  equ     2
    .cpu 68000
    .include sxcall.equ
* X68 GCC Develop
    .even
    .text
    .even
    .globl _foo
_foo:
    link a6,#0
    pea 2.w                 <--- 型が short でもロングワード
    pea 1.w
    jbsr _normal_func       <--- 通常の関数呼び出し
    move.w #2,(sp)          <--- 型が short なのでワード
    pea 1.w
    SXCALL sxcall_func      <--- SXCALL の呼び出し
    unlk a6
    rts
    .even
    .end
```

normal_func は通常の関数、sxcall_func は SXCALL 関数です[7]。通常の関数の呼び出しでは、2 番目の引数がプロトタイプ宣言で「short」

7) SXCALL 関数のほうは、アセンブラソース上でも関数名の前に "_" (アンダーバー) がついていないことに注意してください。これは、自分で sxcall.equ を記述する際に重要です。

と宣言してあっても、実際にはロングワードでスタックに積まれてしまいます。

他方、SXCALL 関数では、プロトタイプ宣言のとおりにワードで積まれています。つまり、新たに SXCALL 関数を宣言する場合、利用したい SX コールの引数のサイズに注意し、ワードで積むべき引数に対しては正しく「short」あるいは「unsigned short」と指定する必要があります。ここで、

バイトサイズの引数はどうするのか

という問題に気づきます。MC68000 ではバイトサイズでスタックに積むことができない[8]ので、SXCALL 宣言で引数の型を「char」にしてもうまくいきません。なかには引数としてバイトサイズを要求する SX コールも存在しますが、バイトサイズの引数は必ず 2 つ続いており、1 つにまとめてワードサイズの引数と見なすことで対処することが可能です。あるいは、ワードサイズの引数の上位バイトと下位バイトにそれぞれ別の意味が割り当てられていると考えるほうが自然かもしれません。

8) バイトサイズでスタックに積んだ場合 (move.b d0,-(sp) など)、スタックは 1 バイトよけいに消費され、スタックは偶数バウンダリに保持されます。

3. A0 レジスタへの返り値

SX コールでは、返り値の受け渡しに D0 と A0 の 2 つのレジスタを使用します。しかし、通常 C 言語の関数の返り値は 1 つですし、アセンブラソース上では D0 レジスタのみを返り値の受け渡しに使用するので、そのままでは A0 レジスタの値を参照することができません。

そこで、X680x0 gcc では、A0 レジスタへの返り値を参照するために、_SXCALLPtr という名前の変数が予約されています。これは、「プログラムの任意の位置での A0 レジスタの値を保持する *void** 型の変数」として実装されており、SXCALL 関数を呼び出した直後に参照することで A0 レジスタの値を利用することができます。

この _SXCALLPtr は「任意の位置」で参照可能ですが、SXCALL 関数を呼び出さずに参照したり、不必要に参照を遅らせたりした場合には、その値に意味がない、出力コードの最適化に悪影響を与えるといった結果を引き起こしますので注意してください。

以上で、SXCALL 関数の仕組みがほぼ理解できたことと思います。

この機能を利用した sxcall.h および sxcall.equ は、すでに『C マガジン』(ソフトバンク刊) の付録フロッピーディスクや、パソコン通信などで配布されていますので、それらを入手すれば簡単に SXCALL 関数を利用できます。

すでに便利なヘッダが存在し、かつ配布されているにもかかわらず、新たに“callno header”を作成した背景には次のような理由があります。これらは裏を返せば SXCALL 関数の機能が不十分な点でもあります。

(1) アセンブラの引数の並びとC言語関数の引数の並びとが必ずしも同じではない。

(2) ライブラリ内部でSXコール呼び出し以外の処理を行っているものがある。

(3) _SXCALLPtr を直接ソース中に記述するのは、特定の環境に依存しすぎる。

これら3つは相互に非常に密接な関係にあります。そこで CMFind 関数を例にとり、まとめて説明します。

```
$A299        CMFind

引数
    long    pt                 ; ポイント（ローカル座標）
    long    winPtr             ; ウィンドウレコードへのポインタ
返り値
    D0.L    =0     該当するコントロールはない
            ≠0     パートコード（下位ワード）
                   /リザルトコード
    A0.L    コントロールレコードへのハンドル
Cの関数
    int     CMFind( LPoint pt, Window *winPtr,
                    Control ***ctrlHdl);
```

CMFind 関数は D0 レジスタにパートコード、A0 レジスタにコントロールレコードのハンドルを返してきます。どちらの返り値も必要なものなので、C言語のプロトタイプ宣言では、3番目の新たな引数として A0 レジスタの値を代入するためのハンドルへのポインタが追加されています。この3番目の引数は、アセンブラでの引数の並びにはありません。

この CMFind 関数を SXCALL 関数として宣言する場合、どのようにすればよいのでしょうか。

```
SXCALL int CMFind (LPoint, Window*);
```

アセンブラにあわせるとこのようになります。

しかし、Workroom では引数が3つあるものが正しく、これではプロ

トタイプ宣言と実際の呼び出しとで引数の並びが違ってしまい、エラーとなります。もちろん、ソースのほうを引数が 2 つの形にし、さらに _SXCALLPtr を参照するように書き換えればよいのですが、すると今度は Workroom でコンパイルできなくなってしまいます。

```
SXCALL int CMFind (LPoint, Window*, Control***);
```

上の例では Workroom にあわせて引数を 3 つにしています。3 番目の引数は無駄になりますが、これで問題なくコンパイルすることができます。

しかし、できあがった実行ファイルは正しく動作しません。その理由は、3 番目の引数に対して適切な処理を行っていたのは純正ライブラリであって、SX コールではなかったということです。

SXCALL 関数の実体は、単に SX コールを呼び出すだけですから、これでは 3 番目の引数に対して何の処理も行われず、プログラムを正しく動作させるためには、結局、プログラム中に _SXCALLPtr の参照を書き足さなければなりません。

このように SXCALL 関数の機能は非常に強力なのですが、一部の SX コールに対してはちょっと力不足です。そこで、間にワンクッション置くようにしてみます。

List 3-48 ● ワンクッション置いた SXCALL 関数の記述

```
SXCALL int sxcall_CMFind (LPoint, Window*);

static inline
int CMFind (LPoint a, Window* b, Control*** c)
{
  int_stat = sxcall_CMFind (a, b);
  *(c) = (Control**) _SXCALLPtr;
  return stat;
}

int foo (LPoint a, Window* b, Control*** c)
{
  return CMFind (a, b, c);
}
```

このようにアセンブラレベルの SXCALL 関数を別名で宣言し、C 言語レベルの関数をインライン関数として記述します。インライン関数のなかでは、まず、SXCALL 関数を呼び出し、さらに従来はライブラリが行っていた 3 番目の引数に関する処理を行います。このインライン関数を利用した場合、呼び出し部分のアセンブラソースは次のようになります。

List 3-49 ● List3-48 のアセンブラソース

```
* NO_APP
RUNS_HUMAN_VERSION   equ     2
    .cpu 68000
    .include sxcall.equ
* X68 GCC Develop
    .even
    .text
    .even
    .globl _foo
_foo:
    link a6,#0
    move.l 12(a6),-(sp)
    move.l 8(a6),-(sp)
    SXCALL sxcall_CMFind      <--- SXCALL 呼び出し
    move.l 16(a6),a1
    move.l a0,(a1)            <--- 3番目の引数の処理
    unlk a6
    rts
    .even
    .end
```

これで、必要十分な処理が効率よく行われていることがわかります。

ヘッダ内でこのようなインライン関数化をすることで、Workroom の仕様にそって書かれたソースを変更しなくてもコンパイルが可能となり、かつ SXCALL 関数の機能を十分に利用した実行ファイルを作成することができるようになります。

実際には callno header では、次のような命名規則をとっています。

List 3-50 ● callno header での SXCALL 関数の命令規則

```
a) ワンクッション必要な場合
  SXCALL int _sx_a299 (LPoint, Window*);
  static inline
  int CMFind (LPoint a, Window* b, Control*** c)
  {
    int_stat = sxcall_CMFind (a, b);
    *(c) = (Control**) _SXCALLPtr;
    return stat;
  }
b) 必要ない場合
  SXCALL int _sx_a29e (Window*, Region**);
  #define CMDraws _sx_a29e
```

9) これが callno header の名前の由来です。

SXCALL 関数の名前を _sx_callno の形式[9]で定義し、C 言語レベルの関数の名前をインライン関数あるいはマクロ名として定義します。既存の SXCALL 関数の定義を変更したり、新たに SXCALL 関数を追加する場合にも、この命名規則に従うようにしてください。

Workroom ヘッダとの関係

callno header は、Workroom のヘッダにパッチを当て、マクロ名 __SX_INLINE__ が定義されていた場合、このパッチ部分からサブディレクトリ下のインラインヘッダファイルを読み込みます。

以下に、パッチ後の Workroom の各ヘッダで callno header を呼び出す部分を示します。

```
#if (defined __SX_INLINE__ && defined __GNUC__)
#include <sx/?????_i.h>     <---- ヘッダごとに適切なインライン
#include <sx/?????_i.h>     <---- ヘッダを読み込む
#else
#ifdef __PROTO_TYPE
ANSI プロトタイプ宣言
#endif /*__PROTO_TYPE */
KR 形式の宣言
#endif
```

各ヘッダから読み込まれるインラインヘッダの対応表を以下に示します (Table 3-6)。

Table 3-6 ● Workroom ヘッダとそれに対応するインラインヘッダ

Workroom ヘッダ	インラインヘッダ
COLOR.H	Color_i.h
CONSOLE.H	Animation_i.h
	Exception_i.h
	Key_i.h
	Keyboard_i.h
	Mouse_i.h
CONTROL.H	Control_i.h
DIALOG.H	Dialog_i.h
EVENT.H	Event_i.h
MENU.H	Menu_i.h
PRINT.H	Print_i.h

Workroom ヘッダ	インラインヘッダ
RESOURCE.H	Resource_i.h
SXGRAPH.H	Font_i.h
	Graph_i.h
	Semaphore_i.h
SXMEMORY.H	Memory_i.h
TASK.H	Task_i.h
TEXT.H	MTest_i.h
	Text_i.h
VIDEO.H	Video_i.h
WINDOW.H	Subwindow_i.h
	Window_i.h

Workroom ヘッダと旧版ヘッダとの併用

callno header では、Workroom ヘッダおよび旧版のヘッダ (SXDEF.H, SXLIB.H) の両方の環境で利用することが可能です。これは型や構造体の名前に対するマクロ名を用意し、使用するヘッダによって、その値を変えることで実現しています。

これらのマクロ名の処理は sx/sxdecl.h で集中して行われており、Workroom か旧版かの判定は、マクロ名 __SXDEF2_H (Workroom の SXDEF2.H で定義される) が定義されていれば Workroom 環境、そうでなければ旧版環境とみなしています。

各マクロ名と、実際の型や構造体の名前との対応を以下に示します (Table 3-7)。

Table 3-7 ● マクロ名と実際の型・構造体名との対応

マクロ名	Workroom 環境	旧版ヘッダ環境
__BOOL	BOOLEAN	int_
__POINTER	Pointer	pointer
__HANDLE	Handle	handle
__HEAP	Heap	Heap
__BLOCK	Block	Block
__MASTER	Master	Master
__MOUSE	Mouse	MsRec
__MSCSR	MsCsr	TXCsr
__EVENT	Event	event
__LPOINT	LPoint	point_t
__RECT	Rect	rect
__REGION	Region	region
__BITIMG	BitImg	unsigned short

マクロ名	Workroom 環境	旧版ヘッダ環境
__LASCII	LASCII	LASCII*
__GSCRIPT	GScript	gScript
__BITMAP	Bitmap	bitmap
__GRAPH	Graph	graph
__POLYGON	Polygon	polygon
__NPOLY	NPoly	void
__NPOLYENV	NPolyEnv	void
__RECTIMG	RectImg	rectimg
__BITS	Bits	bits
__WINDOW	Window	window
__SUBWIN	Subwin	subWindow
__MENU	Menu	menu
__CONTROL	Control	control
__DIALOG	Dialog	dialog
__DILIST	DIList	dlgIList
__TEDIT	TEdit	tEdit
__TEHIS	TEHis	teHis
__TECOLUMN	TEColumn	teColumn
__TSEVENT	TsEvent	tsevent
__TASK	Task	task
__DRAG	Drag	drag
__SCRAP	Scrap	scrap
__OPENFILE	OpenFile	openfile
__ICSTATE	IcState	icstate
__PRINT	Print	prRec
__PDRVRINFO	PDrvrInfo	prtdInfo
__FONTLIST	FontList	FontList
__KEY	Key	KmRec
__KBOARD	KBoard	KbRec
__TX16	TX16	tx16
__GSONEENV	GSOneEnv	void
__PEN	Pen	void
__SPLTBZ	SpltBz	void
__SPLTBSP	SpltBSp	void
__MTEDIT	MTEdit	void
__TESCRAP	TEScrap	void
__TESTYLE	TEStyle	void
__TELSTYLE	TELStyle	void
__TEOPTION	TEOption	void
__ERRPAT	ErrPat	void
__ERRBTN	ErrBtn	void
__PRDLOG	PrDlog	void
__PALET	Palet	void
__CRGB	CRGB	void
__PICKUP	PickUp	void
__VLIST	VList	void
__RGBSCAN	RGBScan	void
__CPDFINFO	CpdfInfo	void
__GSINFO	GsInfo	void

マクロ名	Workroom 環境	旧版ヘッダ環境
__DPATTERN	DPattern	void
__VINFO	VideoInfo	void
__VSINFO	VideoSampleInfo	void
__CINFO	CInfo	void
__CODFINFO	CODFInfo	void
__TEPAGE	TEPage	void
__HWINDOW	HWindow	void
__DEVSET	DevSet	void

まとめ

従来までの XC と純正ライブラリによる SX-WINDOW アプリケーションのプログラミングは、ライブラリの不具合もさることながら、速度を要求すべき処理においてもコール呼び出しにワンテンポ置かなければならず、足枷をつけられた無理のあるプログラミングスタイルであったといえます。

その後登場した、gcc の SX モードによるプログラミングは、ライブラリの不自由さを多少なりとも吸収してくれたものの、やはり、速度面や新しいコールへの対応の手軽さといった面では改善されたとはいいがたいものがありました。

それに対して、callno header の利用による SXCALL 関数のインライン化という新しい手法によるプログラミングでは、従来のライブラリを経由するコール形式と比べて実行ファイルの速度はかなり向上し、ヘッダを書き足すだけで新しいコールへの対応も簡単に行えるなど、SX-WINDOW プログラミングの柔軟さ、生産性において飛躍的な進歩をとげたといっても過言ではないと自負しています。

すでに SX-WINDOW プログラミングを行っている方にも、これから始めようという方にも callno header は大きな助力となるでしょう。みなさんも callno header を上手に利用して軽快に動作するアプリケーションの作成を目指してください。

第4章 LIBSXC

LIBSXC パッケージは、Human68k上のフリーライブラリである LIBC をベースに、SX-WINDOW 対応のための追加、改良を行ったものです。完全に上位互換性を保っているため、従来の LIBC と置き換えて利用することができます。gcc を用いて SX-WINDOW のアプリケーション開発を目指す人には必携といえます。

LIBSXC とは?

本書で用意したLIBSXCライブラリパッケージは、Human68k上のフリーライブラリであるLIBC[1]をもとに、SX-WINDOW用のアプリケーションを開発するために一部修正を行ったものです。そのため、LIBSXCはLIBCに対して上位互換性を保っており、従来のLIBCと置き換えて利用することが可能です。

1)完全フリーのCライブラリ。NIFTY-Serveなどの大手商用BBSや、Network-SX NGなどの草の根BBSで入手可能です。

なお、LIBSXCはgccのSX-WINDOW拡張機能を利用しているため、XC(コンパイラ)での使用は不可能です。必ずgccを使用してください。

また、本書の添付FDに含まれているLIBSXCライブラリパッケージは、LIBC 1.1.32[2]をもとに作成されています。LIBCがバージョンアップするとともにLIBSXCも更新されていく予定ですが、その過程で本書の記述と食い違う部分が出てくる可能性があります。その場合は、最新版のLIBSXCに含まれるドキュメントを参照するようにしてください。

2)1995年5月現在、最新版のLIBCです。

本節では、本書の添付FDで提供するLIBSXCライブラリパッケージの特徴と構成について述べます。

LIBSXCの特徴

LIBSXCの特徴として以下の点が挙げられます。

○ リエントラントなプログラムを作成できる

LIBSXCを使用すると、手軽にリエントラントなプログラムを作成することができます。通常のLIBCやXCのライブラリを使用した場合、ライブラリ内部の変数はプログラムを共有するタスクを通じて共通になります。これでは、タスクが知らないうちに別のタスクがその変数の値を書き換えてしまう可能性があります。これを防ぐには、大域変数を構造体にまとめるなど、プログラマがよけいな手間をかけなければなりませんでした。

LIBSXCでは、基本的にすべてのライブラリ内部変数をタスクごとに独立して持つようにしていますので、手軽にリエントラントな

プログラムを実現することができます。

○ クリーナやアイコン表示に対応

SX-WINDOW には、クリーナやアイコン表示といった Human68k にはなかった機能があります。そのため、ファイルの作成やオープンといったファイル操作を行う場合、SX コール[3]を使用しないと、ファイル操作が行われたというイベントが通知されず、不具合が発生する可能性があります。LIBSXC では、これらのファイル操作専用の SX コールを使用しています。

3) TSOpen や TSCreate など。ファイル操作は主にタスクマンを通して行います。

○ 複数のヒープ領域を扱うことが可能

LIBSXC では、malloc 等のヒープ処理関数を SX-WINDOW 用に変更してあります。SX-WINDOW はマルチタスクですので、同時に複数のタスクが動作しています。しかし、メモリ空間は1つしかないので、複数のタスクがその中でひしめきあうことになります。LIBC や XC のライブラリでは、ヒープ領域はあらかじめライブラリが設定したアドレスから移動せず、必要に応じてメモリの上位の方向に拡大していきます。このようなヒープの拡大方法だと、メモリが余っているのにもかかわらず、ヒープ領域を拡大できない状況が発生します。LIBSXC は、この問題を回避するために、複数のヒープ領域を管理するように変更しています(詳しくは次ページのコラム「SX-WINDOW のメモリ管理と malloc」を参照してください)。LIBSXC は、メモリ空間をより有効に使うことができるわけです。

LIBSXCライブラリの構成

本書で用意した LIBSXC ライブラリパッケージは、LIBC に対して上位互換性を保っています。ここでは LIBSXC のヘッダとライブラリの内容について簡単に紹介しながら、どのようにして上位互換を実現しているかについて説明します。

⊙コラム：SX–WINDOW のメモリ管理と malloc

SX–WINDOW はマルチタスクですので、同時に複数のタスクが動作しています。しかし、メモリ空間は1つしかないので、Fig. 4-1 のように、複数のタスクがその中でひしめきあうことになります。

ここで、malloc 関数を使うことを考えてみます。LIBC や XC のライブラリの malloc 関数は、ヒープ領域からメモリを確保します。

ヒープ領域とは、プログラムがスタートするときに malloc 等のためにあらかじめ確保されるメモリのことです。ヒープ領域はある程度余裕をもって確保されますが、当然、足りなくなることもあります。その場合、実行を中断するか、ヒープ領域を拡大します。

LIBC や XC のライブラリでは、ヒープ領域はあらかじめライブラリが設定したアドレスから移動せず、必要に応じてメモリの上位の方向に拡大していきます。Human68k のようにシングルタスクの OS の場合はこれで十分でしたが、SX–WINDOW のようにマルチタスクで動作する場合には問題が生じます。

たとえば、Fig. 4-2のような状況では、タスク 1 やタスク 3 はヒープ領域を拡大していくことができますが、タスク 2 はすぐあとにタスク 3 がいるため、まだほかの部分にメモリが余っているのにもかかわらず、これ以上、ヒープ領域を拡大することができません。

そこで LIBSXC では、分断された複数のヒープ領域を管理できるようにしました。Fig. 4-2 のような状況で、タスク 2 のヒープ領域を拡大する場合、Fig. 4-1 のようにメモリの余っている箇所にもう 1 つのヒープ領域を確保します。つまり、タスク 2 はヒープ領域を 2 つ持つことになります。

Fig. 4-1 ● LIBSXC の場合

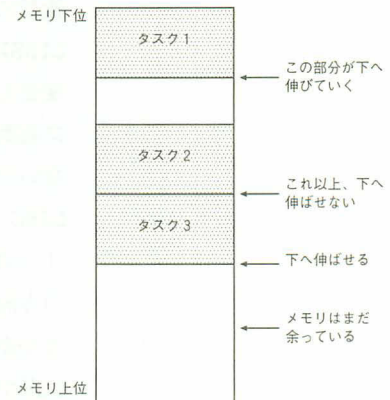
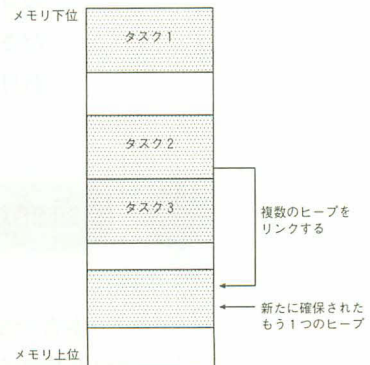



Fig. 4-2 ● LIBC の場合

LIBSXC のヘッダ

LIBSXC ライブラリパッケージに含まれているヘッダと内容は LIBC とまったく同じです。ただし、$include/sys ディレクトリにある次の 3

つのファイルは、LIBSXC と LIBC とでヘッダを共有するために追加あるいは修正が加えられています。

Table 4-1 ● LIBSXC と LIBC で異なるヘッダファイル

ヘッダ	内　容
xstart.h	LIBC との互換性を保つためにあり、gcc の動作モードに応じて下の 2 つのうち、いずれかを読み込む。
xstart_hu.h	Human68k 用の LIBC として動作する場合に読み込まれる。
xstart_sx.h	SX-WINDOW 用の LIBSXC として動作する場合に読み込まれる。

これらはライブラリの内部動作に深く関係したヘッダで、通常、ユーザが直接参照する必要はありません。また、参照した場合には、ANSI C の規格から外れることになります。

LIBSXC と LIBC とでヘッダを共有するために、gcc が自動で定義するマクロ名を利用しています。gcc で SX モードを指定する[4]と、マクロ名 SX_GCC および __SX_GCC__ が自動的に定義されます。この 2 つのマクロ名[5]は、「今、コンパイラは SX モードで動作している」ということを示しています。ヘッダ内では、#ifdef でこれらのマクロ名をチェックし、自動的に LIBSXC と LIBC とを切り替えています。

4) "gcc -SX" で指定します。

5) 必ず 2 つ同時に定義されるので、どちらか一方をチェックすれば十分です。

LIBSXC のライブラリ

LIBSXC ライブラリパッケージには、14 個のライブラリファイルがあります。そのうち libsxc.a(l) は LIBSXC 専用、libc.a(l) は LIBC 専用ですが、それ以外の 12 個のライブラリファイルは LIBSXC と LIBC とで共用されます。

gcc は、動作モードによって libc.a(l) と libsxc.a(l) のどちらをリンクするかを自動的に決定し、リンカに指示を出します。通常の動作モードでは、libc.a(l) をリンクするように指示を出します。一方、SX モードでは、かわりに libsxc.a(l) をリンクするように指示を出します。このように動作モードにあったライブラリが自動的に選択されるので、通常の使用では特に気にする必要はありません。

次に、LIBSXC と LIBC とで共用される 12 個のライブラリファイルについて説明します。なお、これらのライブラリファイルのなかには SX モードでは使用できない (リンクしても正常に動作しない) ものがあります。

それぞれのライブラリファイルについて少し詳しく見てみましょう。

○ そのまま利用可能なもの

libcplus : C++で使用されるライブラリ
libgnu : 四則演算などの基本的なライブラリ
libmb : マルチバイト文字用のライブラリ
libw : ワイド文字用のライブラリ

これらはLIBSXCでも問題なく利用可能です。

○ 利用可能だが、制限もしくは副作用があるもの

libsuper : スーパバイザモードで実行する場合に必要なライブラリ
libtz : タイムゾーン関係のライブラリ

これらはLIBSXCでも明示的にリンクすることで利用することが可能です。ただし、動作上の制限や副作用があります。詳細は、このあとの「4.2 LIBSXCプログラミング」(p.134)で説明します。

○ 関数の一部が使用不可能なもの

libdos : DOSコール用のライブラリ
libiocs : IOCSコール用のライブラリ
libscsi : SCSIコール用のライブラリ

これらはLIBSXCでも明示的にリンクすることで利用可能です。ただし、SX-WINDOW上では使用してはいけない関数を含んでいるので、使用する場合は注意が必要です。詳細は、このあとの「4.2 LIBSXCプログラミング」(p.134)で説明します。

○ 現時点では使用不可能なもの

libprof : プロファイラ(計測)用のライブラリ
libsignal : シグナル関係のライブラリ

これらはSX-WINDOWのことをまったく考慮していないので、使用することができません。プロファイラはあると便利なので、将来的にはこれに対応したいと考えています[6)]。

6)gccは、SXモードではプロファイラ使用のオプションが無視されます。

○ 使用不可能なもの

libbas : X-BASIC関係のライブラリ

X-BASICはSX-WINDOWに対応していないので、libbasを使用することはできません。

これら12個のライブラリを利用する場合、gccに対して明示的にリンクを指示しなければなりません。ただし、libgnuだけは必ずリンクされるので、明示的に指示する必要はありません。たとえば、libdosと

libmb をリンクしたい場合、gcc に対して次のようにコマンドラインから指示します。

```
gcc -SX foo.c -ldos -lmb
```

“-l” は「ライブラリをリンクしろ」というオプションで、そのあとにリンクしたいライブラリの名前を続けて書きます。ライブラリの名前には、“libdos.a” や “libmb.a” から先頭の “lib” と拡張子を除いた、“dos”, “mb” といった文字列を指定します。

LIBSXC プログラミング

ここではLIBSXCを使った実際のプログラミングについて説明します。

LIBSXCはLIBCをもとに作成されており、LIBCとまったく同じ感覚でプログラムを作成することができます。むしろgccのSX拡張機能のおかげで、XCを使用したときのように大域変数を構造体にまとめたりする手間がなく、すっきりとしたプログラムになります。

そこで、まず基本的なLIBSXCの使用法を習得するために、第3章「SX-WINDOWS ver.3.1 開発キット」（p.73）で紹介したサンプルプログラムをLIBSXC用に修正してみます。実際にプログラムを修正してみることで、LIBSXCへの移行が非常に簡単なものであることを感じとってください。

次に、よく遭遇するトラブルについて、その原因と対処法を説明します。ここで取り上げるトラブルは、パソコン通信を通じてユーザから寄せられる質問のなかでも特に多いものです。

そして最後にLIBSXC独自のプログラミングテクニックについてふれます。これらのテクニックは中級以上のSX-WINDOWプログラマの必修事項といえるでしょう。

基本的なLIBSXCプログラミング

まず基本的なLIBSXCの使用法を習得するために、「CGA再生のサンプル」（p.109）で紹介したサンプルプログラムVMSMPL2.Cを、LIBSXC用に修正してみましょう。

とりあえず動かす

きっちりとLIBSXC用に修正する前に、「とりあえず動く」ように最低限の修正をしてみましょう。修正するのはたった1行です。

VMSMPL2.Cの56行目から始まる、「カーネルの設定」という箇所の、

List 4-1 ● VMSMPL2.C 62 行目 (修正前)

```
62: char _sxkernelcomm[]= "sxkernel.x";
```

という行の先頭に "common" を付け足します。

List 4-2 ● VMSMPL2.C 62 行目 (修正後)

```
62: common char _sxkernelcomm[]= "sxkernel.x";
```

とりあえず動かすための修正はこれだけです。実に簡単なものです。本当に動くかどうかコンパイルして実行してみましょう。

```
A:> gcc -SX -O vmsmpl2.c -lsxc -lsx3 -lsx
```

きちんと動きましたか？ Workroom に付属しているサンプルプログラムも、すべてこの修正で LIBSXC 用に (とりあえず) 変更することができます。

ところで、サンプルプログラム VMSMPL2.C をよく見ると、先ほどの「カーネルの設定」という部分は次のようになっています。

List 4-3 ● VMSMPL2.C 56 行目付近

```
56: /*
57:  * カーネルの設定
58: */
59: #ifdef __SX_GCC__
60: common char _sxkernelcomm[]= "sxkernel.x";
61: #else
62: char _sxkernelcomm[]= "sxkernel.x";
63: #endif
```

これはマクロ名 __SX_GCC__ [1]をテストすることで、SX モードのときと、そうでないときとに場合分けしています。このように記述することで、XC と gcc + LIBSXC のどちらでもコンパイルできるようになります。本書の添付 FD に収録されているサンプルは、SXCSMPL.C を除いてすべてこのように LIBSXC でもコンパイルできるようになっています。

1) 「LIBSXC のヘッダ」(p.130) を参照。

きちんと修正する

さて、先ほどの修正でとりあえず動くようにはなりました。しかし、

これだけでは本当に「ただ動く」だけで、gcc + LIBSXC を使う意味が半減してしまいます。

gcc + LIBSXC を使う大きな利点の 1 つは、大域変数を素直[2]に記述できることにあります。XC を使った場合、Workroom のサンプルプログラムや本書の第 3 章「SX–WINDOWS ver.3.1 開発キット」 (p.73) のサンプルプログラムのように、大域変数を構造体でひとまとめにし、さらに大域変数にアクセスする関数すべてに大域変数へのポインタを引数として渡す必要がありました (List 4-4)。

2) ここでいう「素直」とは、「ごく普通に」くらいの意味です。

List 4-4 ● 大域変数をまとめた構造体と関数プロトタイプ

```
 1: /*                                /*
 2:  * タスク毎に確保する変数の構造体      * プロトタイプ
 3:  */                               */
 4: typedef struct {                 int CheckGRW (glVal*);
 5:   int EventMask;                 int init (glVal*);
 6:   TsEvent EventRec;              void tini (int, glVal*);
 7:       (中略)                              (中略)
 8:   LPoint AnimSize;               void GetDrag (glVal*);
 9:   int AnimMode;                  void SystemEvent (glVal*);
10: } glVal;
```

なぜ、このような手間をかける必要があるのでしょうか？

XC で普通に大域変数を使用した場合、その大域変数はタスク共通の変数になってしまいます。つまり、ただ “TsEvent EventRec;” とすると、プログラムを共有する他のタスクがこの変数を書き換えてしまう可能性が出てきます。それでは困るので、XC では大域変数を構造体にまとめ、実行時に auto 変数として、あるいはヒープ上に確保することで大域変数をタスクごとに分離しています。

大域変数を構造体にまとめるのは悪くはない[3]のですが、実際に大域変数にアクセスする際、“gv->EventMask” といちいち書かなければなりません[4]。

一番問題なのは、大域変数にアクセスする関数すべてに大域変数へのポインタを引数として渡さなければならないことです。関数の引数は関数を呼び出すたびにスタックに積まれるので、プログラム実行時にかなりのオーバーヘッドになります。

さて、このような手間も、gcc + LIBSXC では無用となります。gcc の SX モードでは素直に “TsEvent EventRec;” と大域変数を記述するだけでタスクごとに分離した大域変数になり、プログラムを共有する別

3) 変数の名前空間を分離させることができるので、まとめること自体は非常によいアイディアです。

4) “gEventMask” と書くのと大差ないといえばそうなのですが…。

のタスクがこの変数を書き換えるようなことはなくなります。先ほどの大域変数と関数プロトタイプは次のように修正します (List 4-5)。

List 4-5 ● LIBSXC 用に修正後の大域変数と関数プロトタイプ

```
1: /*                              /*
2:  * タスク毎に確保する変数の構造体    * プロトタイプ
3:  */                              */
4: int gEventMask;                 int CheckGRW (void);
5: TsEvent gEventRec;              int init (void);
6: Window* gWinPtr;                void tini (int);
7:      (中略)                          (中略)
8: LPoint gAnimSize;               void GetDrag (void);
9: int gAnimMode;                  void SystemEvent (void);
```

大域変数が素直に記述され、関数の引数にもよけいなポインタがありません。大域変数の名前の先頭に "g" を追加してあるのは、「大域変数である」ことを明示するためのもので、特に他の変数との衝突が起こらなければつける必要はありません。

次に個々の関数も修正しましょう。ここでは例として KeyDownEvent 関数を取り上げます。

List 4-6 ● 修正前の KeyDownEvent 関数

```
1: void
2: KeyDownEvent (glVal* gv)
3: {
4:   /*イベントを取り除く*/
5:   TSGetEvent (gv->EventMask, &gv->EventRec);
6: }
```

List 4-7 ● 修正後の KeyDownEvent 関数

```
1: void
2: KeyDownEvent (void)
3: {
4:   /*イベントを取り除く*/
5:   TSGetEvent (gEventMask, &gEventRec);
6: }
```

行うべき修正は 2 つです。1 つは関数の引数から大域変数へのポインタを削除すること、もう 1 つは実際に大域変数をアクセスしている部分

を書き換えます。

KeyDownEvent 関数の場合、まず、関数の引数から大域変数へのポインタ "glVal* gv" を削除します。削除した結果、引数がなくなるときは引数を "void" にしておきます[5]。次に、実際に大域変数にアクセスしている部分、"gv->EventMask", "gv->EventRec" をそれぞれ "gEventMask", "gEventRec" と修正します。

他の関数についても同じ手順で修正を加えます。全部修正し終えたら、先ほどと同様にコンパイルして実行してみましょう。きちんと動きましたか？ 動かないときは、もう一度ソースを確認してみましょう。きっと修正し忘れや、修正間違いがあるはずです。それでも駄目なときは、このあとの「4-3 トラブルシューティング」(p.139) を参照してみてください。

5) ANSI C では、引数のない関数は引数に void を書くことになっています。

まとめ

既存のプログラムの修正を通して、LIBSXC でのプログラミングの実際を紹介してきました。LIBSXC を使うのは全然難しくなく、実は使ったほうがプログラミングが楽になるんだ、ということを漠然とでも感じることができればしめたものです。

あとは「習うより慣れろ」で、1つでも多くのプログラムを作って、順次ステップアップしていってください。その過程でわからないことがあったときには、本書の APPENDIX B 章「LIBSXC 便利帳」(p.159) を開いてみてください。

トラブルシューティング

ここではLIBSXCを使用していて比較的よく遭遇する2つのトラブルについて、その原因と対処方法を説明します。ここで取り上げるトラブルは、パソコン通信を通じてユーザから寄せられる質問のなかでも特に多いものです。

個々のトラブルを説明する前に、いくつか注意点を述べておきます。

まず、ここで取り上げる2つのトラブルはいずれもリンクの段階で起きるものです。そのため、リンカに関する知識が必要になります。また、具体的にトラブルを見つける段階ではアセンブラの知識が必要になります。予備知識なしでもわかるように詳しく説明はしますが、最低限MC68000のアセンブリ言語が理解できることを前提とします。

Relative error

プログラムが大きくなってくると、管理の都合やコンパイル時間の短縮のため、プログラムを複数に分割することがあります。“Relative error”は、このように複数に分割されたプログラムをリンクする段階で発生します。次の小さなプログラムを例にとり、エラーの原因とその対処方法を見ていきます。

List 4-8 ● プログラム foo.c

```
1:  common int foo = 1;
```

List 4-9 ● プログラム bar.c

```
1:  extern int foo;
2:
3:  int bar (void)
4:  {
5:    return foo;
6:  }
```

foo.c のほうでは変数 foo が初期値 1 として定義されており、bar.c から変数 foo を参照しています。

見ればすぐにわかるように、foo.c では "common int foo" となっているのに、bar.c では "int foo" となっていて "common" がありません。これでは変数の記憶クラスが違うため、bar.c からの変数 foo の参照はうまくいきません。

この 2 つのプログラム foo.c と bar.c を gcc の SX モードでコンパイル、リンクしてみます。

```
A:> gcc -c -SX foo.c bar.c
A:> gcc -SX foo.o bar.o
X68k SILK Hi-Speed Linker v2.29 Copyright 1989-93 SALT
Relative error in bar.o
 at 00000006 (text)
A:/usr/bin/gcc.x: Program lk.x exit status 65535.
```

"Relative error" はリンカ hlk の出すエラーメッセージで、相対アドレッシングで届かないアドレスを指定した場合や、アドレスを示す属性のシンボルをワードやバイトの値として書き込もうとすると発生します。"in bar.o at 00000006 (text)" というのは、オブジェクトファイル bar.o 中のテキストセクションの 00000006 番地 (16 進数) でエラーが発生したことを示しています。

さて、エラーの発生した場所はわかりましたが、その原因はこれだけではわかりません。

そこで、アセンブラ has のオプション '*-p*' を使ってリストファイルを作成します。リストファイルとはアセンブルリストのことで、List 4-10, List 4-11 のように、アセンブリ言語の命令とそれに対応する機械語のコード (単なる 16 進数) が対になったものです。このリストファイルは通常出力されないので、必要な場合にはアセンブラ has に対してオプション '*-p*' を指定する必要があります。直接 has を呼ぶ場合には "has -p" とすればよいのですが、今回のように has が gcc から間接的に呼び出される場合、環境変数 HAS [1] を用いて間接的に指定します。

1) has がつねに使用するオプションを設定します。has は実行時に環境変数の内容をコマンドラインの最後に追加します。

```
A:> set HAS -p
A:> gcc -c -SX foo.c bar.c
```

このようにして得られたリストファイル foo.prn, bar.prn のうち、必要な部分のみを以下に示します。

List 4-10 ● foo.prn: foo.c のリストファイル

```
 1:   5 00000000                * X68 GCC Develop
 2:   6 00000000                        .globl _foo
 3:   7 00000000                        .even
 4:   8 00000000                        .data
 5:   9 00000000                        .even
 6:  10 00000000                _foo:
 7:  11 00000000 00000001               .dc.l 1  <--- 変数 foo の実体
 8:  12 00000004                        .even
 9:  13 00000004                        .end
```

List 4-11 ● bar.prn: bar.c のリストファイル

```
 1:   5 00000000                      * X68 GCC Develop
 2:   6 00000000                              .even
 3:   7 00000000                              .text
 4:   8 00000000                              .even
 5:   9 00000000                              .globl _bar
 6:  10 00000000                      _bar:
 7:  11 00000000 4E560000                     link a6,#0
 8:  12 00000004 202D????                     move.l _foo.w(a5),d0
 9:                  ~~~~ オフセットは 2 バイト分しかない
10:  13 00000008                              jbra ?1
11:  14 00000008                      ?1:
12:  15 00000008 4E5E                         unlk a6
13:  16 0000000A 4E75                         rts
14:  17 0000000C                              .even
15:  18 0000000C                              .end
```

変数 foo の実体がある foo.prn のほうでは、変数 foo を指すラベル _foo に 4 バイトの値が与えられています。しかし、bar.prn ではラベル _foo を参照している部分に 2 バイト分の領域しかありません。そのためにリンクができず、“Relative error” となります。

“Relative error” が発生するのは、多くの場合、この例にあるような記憶クラスの違いが原因です。リストファイルを見て、ラベル _foo の参照が原因だとわかったら、次は個々のソースファイルから変数 foo を宣言している箇所を探し、記憶クラスがあっているかどうかを確認してください。

この例の場合、foo.c を “int foo = 1;” とするか、あるいは bar.c を “extern common int foo;” として記憶クラスをあわせます。

ここで 1 つ注意しておきます。リンカ hlk のマニュアルの「トラブル

シューティング」に、「SX-WINDOW の OBJR 形式のプログラムの場合は、さらに -fall-remote もつけてみて下さい」という記述があります。gcc に -fall-remote を指定すると、通常の変数の記憶クラスが remote になります。

先ほどの例を、記憶クラスがあっていない状態で "-fall-remote" をつけてコンパイルしてみます。foo.c のほうの結果は変わりません。

List 4-12 ● bar.prn: -fall-remote をつけたときのリストファイル

```
 1:  5 00000000                    * X68 GCC Develop
 2:  6 00000000                              .even
 3:  7 00000000                              .text
 4:  8 00000000                              .even
 5:  9 00000000                              .globl _bar
 6: 10 00000000                    _bar:
 7: 11 00000000 4E560000                     link a6,#0
 8: 12 00000004 41F9????????                 lea _foo,a0
 9:                  ~~~~~~~~ オフセットは 4 バイト分ある
10: 13 0000000A 2030D800                     move.l (a0,a5.l),d0
11: 14 0000000E 4E5E                         unlk a6
12: 15 00000010 4E75                         rts
13: 16 00000012                              .even
14: 17 00000012                              .end
```

こうすると、ラベル _foo を参照している部分に 4 バイトの領域があるので、リンクすることができます。しかし、リンクできるだけで記憶クラスがあっていないことには変わりがなく、できあがったプログラムは正常に動作しません。

"Relative error" が発生したからといって、安易に "-fall-remote" をつけて安心してはいけません。きちんとエラーの原因を調べるようにしましょう。面倒なようですが、それがプログラミング技術の向上にもつながります。

● Over flow error

プログラム中で大きな大域変数を確保した場合、変数参照のためのオフセットが 2 バイトに収まらなくなり、リンカで "Over flow error" が発生することがあります。ここでも次の小さなプログラムを例にとって、エラーの原因とその対処方法を見ていきます。

List 4-13 ● プログラム fubar.c

```
 1:  char _bar[68000];
 2:  char _foo[68030];
 3:
 4:  char* foo (void)
 5:  {
 6:    return _foo;
 7:  }
 8:
 9:  char* bar (void)
10:  {
11:    return _bar;
12:  }
```

このプログラム fubar.c を gcc の SX モードでコンパイル、リンクしてみます。

```
A:> gcc -SX -c fubar.c
A:> gcc -SX fubar.o
X68k SILK Hi-Speed Linker v2.29 Copyright 1989-93 SALT
Over flow in fubar.o
 at 00000014 (text)
A:/usr/bin/gcc.x: Program lk.x exit status 65535.
```

“Over flow” はリンカ hlk の出すエラーメッセージで、シンボルの値を書き込もうとしたときに、2 バイトもしくは 4 バイトで表現できる値の範囲を超えた場合に発生します。“in fubar.o at 00000014 (text)” というのは、オブジェクトファイル fubar.o 中のテキストセクションの 00000014 番地 (16 進数) でエラーが発生したことを示しています。

“Relative error” のときと同様、まず、リストファイルを調べます。

```
A:> set HAS -p
A:> gcc -c -SX fubar.c
```

List 4-14 ● fubar.prn: fubar.c のリストファイル

```
1:   5 00000000                   * X68 GCC Develop
2:   6 00000000                           .even
3:   7 00000000                           .text
4:   8 00000000                           .even
5:   9 00000000                           .globl _foo
6:  10 00000000                   _foo:
7:  11 00000000 4E560000                  link a6,#0
8:  12 00000004 41ED????                  lea __foo.w(a5),a0
```

```
 9: 13 00000008 2008                     move.l a0,d0
10: 14 0000000A 4E5E                     unlk a6
11: 15 0000000C 4E75                     rts
12: 16 0000000E                          .even
13: 17 0000000E                          .globl _bar
14: 18 0000000E                   _bar:
15: 19 0000000E 4E560000                 link a6,#0
16: 20 00000012 41ED????                 lea __bar.w(a5),a0
17:                         ~~~~ オフセットは2バイト分しかない
18: 21 00000016 2008                     move.l a0,d0
19: 22 00000018 4E5E                     unlk a6
20: 23 0000001A 4E75                     rts
21: 24 0000001C                          .even
22: 25 0000001C                          .rbss
23: 26 00000000                          .xdef __foo
24: 27 00000000                          .rcomm __foo,68030
25: 28 00000000                          .xdef __bar
26: 29 00000000                          .rcomm __bar,68000
27: 30 00000000                          .even
28: 31 00000000                          .end
```

fubar.c では _foo と _bar という64Kバイトを超える変数が2つ確保されています。そのため、関数 foo での _foo の参照はうまくいきますが、次の関数 bar での _bar の参照は _foo が64Kバイトを超える大きさをもっているので、2バイトのオフセットでは足りなくなり、うまくいきません。そのため、リンカで“Over flow”が発生します。

これは“Relative error”のときのように記憶クラスの違いが原因ではなく、純粋にオフセットが足りないことが原因です。そのため“-fall-remote”を使うことが正しい解決方法の1つとなります。“-fall-remote”をつけてコンパイルし直すと、リストファイルは次のようになります。

List 4-15 ● fubar.prn: -fall-remote をつけたときのリストファイル

```
 1:  5 00000000                   * X68 GCC Develop
 2:  6 00000000                          .even
 3:  7 00000000                          .text
 4:  8 00000000                          .even
 5:  9 00000000                          .globl _foo
 6: 10 00000000                   _foo:
 7: 11 00000000 4E560000                 link a6,#0
 8: 12 00000004 200D                     move.l a5,d0
 9: 13 00000006 D0BC????????             add.l #__foo,d0
10: 14 0000000C 4E5E                     unlk a6
11: 15 0000000E 4E75                     rts
12: 16 00000010                          .even
```

```
13:  17 00000010                              .globl _bar
14:  18 00000010                     _bar:
15:  19 00000010 4E560000                     link a6,#0
16:  20 00000014 200D                         move.l a5,d0
17:  21 00000016 D0BC????????                 add.l #__bar,d0
18:                  ~~~~~~~~ オフセットは 4 バイト分ある
19:  22 0000001C 4E5E                         unlk a6
20:  23 0000001E 4E75                         rts
21:  24 00000020                              .even
22:  25 00000020                              .rlbss
23:  26 00000000                              .xdef __foo
24:  27 00000000                              .rlcomm __foo,68030
25:  28 00000000                              .xdef __bar
26:  29 00000000                              .rlcomm __bar,68000
27:  30 00000000                              .even
28:  31 00000000                              .end
```

もう1つの解決方法は、大きなメモリを実行時に動的に確保する方法です。

具体的には malloc 関数を利用し、次の fubar2.c のようにします。

List 4-16 ● プログラム fubar2.c

```
 1:  char* _bar;
 2:  char* _foo;
 3:
 4:  char* foo (void)
 5:  {
 6:    return _foo;
 7:  }
 8:
 9:  char* bar (void)
10:  {
11:    return _bar;
12:  }
13:
14:  void init (void)
15:  {
16:    _bar = malloc (68000);
17:    _foo = malloc (68030);
18:  }
```

この場合、関数 foo や bar を呼ぶ前に必ず関数 init を呼んで変数 _bar, _foo を初期化する必要があります[2)]。

2) 実際には malloc の返り値をチェックする必要もありますが、ここでは省略しました。

まとめ

ここではLIBSXCを使用する際、リンクの段階で比較的よく遭遇するエラーについて説明しました。実際にエラーの原因を探り解決するには、多くの知識となによりも経験を必要とします。筆者もいきなりこのような知識を身につけたわけではなく、数多くの試行錯誤の末、トラブル解決に対するカンのようなものを養ってきました。みなさんもエラーにめげず、「謎解き」とでも思って原因を究明するようにしてください。

LIBSXC 独自のプログラミングテクニック

ここでは「簡単な SX-WINDOW のプログラムなら作成できる」という中級以上のプログラマを対象に、LIBSXC 独自のプログラミングテクニックやトピックについて説明します。

OBJR 型モジュールの作成と注意点

LIBSXC を使用して実行ファイルを作成した場合、デフォルトでは OBJR 型モジュール[1]になります。

通常、プログラムはテキストセクションとデータセクションの２つから構成されています。テキストセクションにはプログラムのコード部分など実行中に変化しないものが、データセクションには変数など実行中に変化するものが含まれます。

OBJR 型モジュールでは、リエントラントという名称のとおり、コードが収められているテキストセクションを共有し、タスクごとにデータセクションなどを用意するという形式で資源の有効利用を実現しています。LIBSXC を使用した OBJR 型モジュールでも、通常の変数[2]はタスクごとのデータセクションに配置されます。しかし、gcc の common 修飾子をつけた変数[3]は、「複数のタスクに共通な変数」として、通常とは別のデータセクションに配置されます。

LIBSXC のライブラリ内部変数[4]の一部は、この common 修飾子をつけた変数として定義され、テキストセクションを共有するタスクの間で共通になっています。そのため、これらのライブラリ内部変数を操作すると、テキストセクションを共有するタスクすべてに影響を与えることに注意しなければなりません。

具体的には、次のような制限や副作用があります。

○ １つのタスクを (コマンドラインで指定して) スーパバイザモードで実行し、別のタスクをユーザモードで実行することはできません。そもそも SX-WINDOW では、一般のアプリケーションがスーパ

1) OBJR 型モジュールの詳細は、本書のコラムや Workroom のマニュアルを参照してください。

2) たとえば、“int a;” など。

3) たとえば、“common int a;” など。

4) ライブラリの実行を制御する特別な変数。

バイザモードで動作することは基本的に禁止しています。

○ OBJR 型モジュールを複数起動した場合、一度タスクが切り替わり、次に制御が戻ってきたときには、errno の内容は保証されません。errno の典型的な使用手順は、

List 4-17 ● eerorno の典型的な使用手順

```
1:  errno = 0;
2:  x = sqrt (y);
3:  if (errno)
4:    {
5:     fprintf (stderr, "sqrt failed.  errno = %d\n",
       errno);
6:     x = 0;
7:    }
```

のように errno をクリアしてからライブラリを呼び出し、その直後で値をチェックします。これは ANSI で規定された用法で、これに従うかぎり、問題は起きません。

○ 同様に、タスクが切り替わったあとの time 関連のライブラリ内部変数の内容は保証されません。必要に応じて別の変数に保存するようにしてください。あとで述べる方法で OBJC 型のモジュールを作成した場合、これらの制限はいっさい生じません。

○ IOCS/DOS/SCSI コール等は一部を除いて使用できません。SX-WINDOW では、IOCS/DOS/SCSI コール等を使って直接画面にアクセスしたり、デバイスを操作することが一部を除いて禁じられています[5]。これを守らなかった場合、SX-WINDOW のクリーナの動作がおかしくなったり、あるいは画面が乱れたりするなど、SX-WINDOW の使用に支障をきたす可能性があります。SX-WINDOW のようなウィンドウシステムでは、Look & Feel のガイドライン[6]を守ることが大切ですから、禁止されたコールは使用しないでください。具体的にどのコールが使用禁止なのかについては Workroom のマニュアルを参照してください。

5) たとえば、ファイルのオープン、クローズには必ず SX コールを使いますが、読み込みは DOS コールを使います。見ためや操作方法を統一するための規約です。

6) SX-WINDOW に関しては、Workroom のマニュアルにその記述があります。

OBJR 型以外のモジュールの作成

LIBSXC を使用して実行ファイルを作成した場合、デフォルトでは OBJR 型のモジュールになります。OBJR 型以外のモジュールを作成したい場合には、モジュールタイプを明示的に指定する必要があります。

たとえば、OBJC 型モジュールを作成したい場合には、

List 4-18 ● objc.s: OBJC 型モジュールの指定

```
1:  .xdef OBJECTTYPE
2:  OBJECTTYPE .equ 'OBJC'
```

といったアセンブラのソースを作成し、他のファイルといっしょにリンクします。

```
gcc -SX foo.c objc.s
```

OBJO 型のモジュールを作成する場合も同様にします。

カーネルの変更

SX-WINDOW 用のアプリケーションは、SX シェル (SXWIN.X) から起動するのが普通ですが、Human68k 上で SX-WINDOW 用のプログラムを開発している場合、デバッグのためにコマンドラインから起動することが多くなります。

コマンドラインから起動した場合、SX-WINDOW のカーネル[7] を起動しなければなりませんが、これは LIBSXC のスタートアップルーチン[8] が処理します。LIBSXC の場合、デフォルトでは "sxkernel" [9] というファイル名のカーネルが実行されます。

たとえば、カーネルとして "sxwdb" [10] を起動し、さらにカーネルに対して "-17" [11] というオプション[12] を指定する場合、

List 4-19 ● kernel.c: デバッグ用カーネルを用いる

```
1:  common const char _sxkernelcomm[] = "sxwdb -17";
```

といった C のソースを作成し、他のファイルといっしょにリンクします。

7) SX-WINDOW 用のアプリケーションの動作に必要な、最低限の初期化を行うプログラムです。

8) main 関数に実行が移る前に、ライブラリ内部変数の初期化等、必要な処理を行うルーチンです。

9) 最も単純なカーネル。

10) SX シェル (SXWIN.X) と同等の環境を提供するデバッグ用カーネルです。実は SXWIN.X を SXWDB.X にリネームしただけのものです。

11) "-17" は、SYSTEM.LB, BUILTIN.LB, ICON.LB をメモリに読み込む、というオプションです。

12) カーネルに対するオプションの詳細は、Workroom のマニュアルを参照してください。

```
gcc -SX foo.c kernel.c
```

アプリケーション開発の初期段階では、起動の速い"sxkernel"を使い、完成に近づくにつれ、実際の環境に近い"sxwdb"を使うようにするとよいでしょう。

メモリの動的確保

LIBSXCでは、LIBCのヒープ関係の関数を改良することでSX-WINDOWの環境への適応を試みています[13]。実際の使用に際しては、普通にmalloc関数やfree関数を用いることができ、次の2点を除いては特に注意すべきことはありません。

○ LIBSXCではLIBCとは違うヒープ管理をしています。起動時オプション"-+-h:"や、ライブラリ内部変数_heapsizeによって指定、確保されるLIBCのヒープ領域は使用されません。
デフォルトでは、このヒープ領域のサイズは0になっています。

○ brk関数や、sbrk関数といった、ヒープを拡張するような関数は使用しないでください。これらの関数はLIBCのヒープ領域を対象としています。LIBSXCでは安全のため、これらの関数が_hsta ~ _last ~ _mmax[14]の間でつねに成功するように変更してありますが、すでに述べたように、使用されない領域をいくら操作しても無意味です。
LIBSXCのヒープ領域は必要に応じて自動的に拡張されるので、brk関数やsbrk関数を使う必要はありません。

次にLIBSXCの扱うヒープ領域の構造について、少し詳しく説明します。ただし、LIBCのヒープ領域の構造[15]を理解しているものと仮定して話を進めます。

LIBSXCでは、Fig. 4-3のように、複数の不連続なヒープ領域がリスト構造[16]でつながっています。各ヒープ領域は、前のヒープ領域へのポインタ[17]と、ヒープ本体とから構成されています。

13) コラム「SX-WINDOWのメモリ管理とmalloc」(p.130)を参照してください。

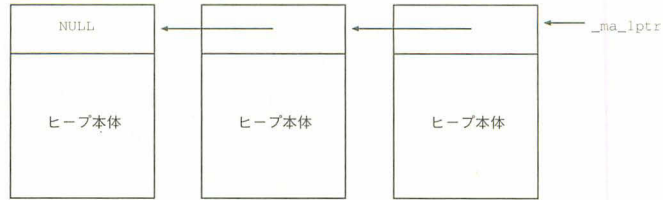
14) LIBCのヒープ領域を指す内部変数です。

15) LIBCではBoundary Tag Methodという方法をもとにしています。Boundary Tag Methodについては、LIBSXCのソースsrc/stdlib/malloc.cに詳細な記述があるので、適宜参照してください。

16) 複数の要素が手をつないで1列に並んでいるようなデータ構造のことです。各要素が手をつないでいるので、先頭から順番に要素をたどっていくことができます。

17) これが「手」に相当します。

Fig. 4-3 ● LIBSXC のヒープ領域

NULL　ヒープ本体　ヒープ本体　ヒープ本体　_ma_lptr

ライブラリ内部変数 _ma_lptr が最後のヒープ領域へのポインタを保持していて、全体として単方向リスト[18]を形成しています。ヒープ本体は LIBC のそれとまったく同じです。

malloc 関数などによってメモリブロックの確保が要求されると、LIBSXC はこれらのヒープ領域から適当な空きブロックを探してきます。ヒープ領域中に空きブロックが見つからない場合は、SX コール MMChPtrNew[19]により新たなヒープ領域を確保し、そこから空きブロックを探します。新たに確保されるヒープ領域のサイズは、ライブラリ内部変数 _SX_GETMEM_SIZE によって指定することができます。

○ size_t _SX_GETMEM_SIZE

新たに確保するヒープ領域のサイズをバイト単位で指定します。デフォルトは 32K (32768) バイトです。ヒープ領域を確保するたびに参照されるので、実行中でも随時変更することが可能です。_SX_GETMEM_SIZE の値を超えるヒープ領域が必要な場合は、その分だけ確保されます。

プログラムによっては、一時的に malloc で大量のメモリを確保したあと、すぐに大半を free にしてしまうことがあります。このような場合、あるヒープ領域全体が完全に未使用状態になってしまうことも考えられます。このようなヒープ領域をそのままにしておくとメモリの無駄になるので、LIBSXC では適当なタイミングで空のヒープ領域を削除します。実際に削除が行われるのは、free が実行されたときです。また、プログラム終了時には、すべてのヒープ領域が自動的に削除されます。

そのほかの LIBSXC のヒープ管理に関する詳細は、LIBSXC ソースコードパッケージ中の、以下のソースファイルを参照してください。

○ src/stdlib/_ma_new.c

○ src/sxstartup/_exit.c

○ _getmem_size.c

18) おのおののポインタが 1 つ前の領域のみを指し、要素を後ろから前にしかたどっていけないので「単方向リスト」といいます。ほかに双方向リストや環状リストがあります。

19) つまり、おのおののヒープ領域は再配置不可能なブロックです。

○ `_sx_killheap.c`

本節の最初でも述べたように、LIBSXC の実際の使用に際しては、ごく普通に malloc 関数や free 関数を用いることができるように設計してあります。あまり難しく考えず、気軽に使用してください。

カーネルの変更やモジュールタイプの変更は、SX-WINDOW に関するある程度の知識を必要とします。自分が行う変更によって何が変わるのかを正しく理解してから使うようにしてください。

APPENDIX

SX31KIT 構造体の定義

SX31KIT で新たに追加された構造体名を Table A-1 に示します。

Table A-1 ● SX31KIT で追加された構造体

構造体名	ヘッダ名	内容
CRGB	COLOR.H	RGB レコード
CInfo	COLOR.H	カラーレコード
Palet	COLOR.H	パレットレコード
pltlList	COLOR.H	パレットテンプレート
cPick	COLOR.H	カラーピックアップルーチン用
ErrPat	DIALOG.H	ユーザ定義アニメーション
ErrBtn	DIALOG.H	ユーザ定義ボタン
PrDl	PRINT.H	プリンタダイアログレコード
GraphGlobal	SXGRAPH.H	グラフマングローバルワーク
CpdfInfo	SXGRAPH.H	CPDF リソース情報
GsInfo	SXGRAPH.H	GScript 情報
DPatPac	SXGRAPH.H	ディザパターンのアイテム情報
DPattern	SXGRAPH.H	ディザパターン情報
CODFInfo	TASK.H	CODF リソース情報
COIF	TASK.H	COIF リソース情報
cCell	TASK.H	圧縮されたセルのデータ構造
MTEdit	TEXT.H	マルチフォントテキストエディットレコード
TEScrap	TEXT.H	テキストエディットスクラップレコード
TEStyle	TEXT.H	スタイル情報レコード
TELStyle	TEXT.H	文字サイズ付きスタイル情報レコード
TEOption	TEXT.H	オプション情報レコード
TEPage	TEXT.H	ページ情報レコード
VideoInfo	VIDEO.H	ビデオ情報レコード
VideoSampleInfo	VIDEO.H	ビデオサンプル情報レコード
HWindow	WINDOW.H	階層ウィンドウレコード

マクロの定義

SX31KIT で定義されるマクロ名とその役割を Table A-2 に示します。

Table A-2 ● SX31KIT で定義されるマクロ名

マクロ名	ヘッダ名	役割
__SX31KIT__	SXDEF2.H	SX31KIT のヘッダを用いていることを示します。現在は「1」が定義されています。
SXVER2	TASK.H	Workroom と同様に、動作可能な SX システムのバージョン 0x0201 が定義されています。
SXVER3	TASK.H	動作可能な SX システムのバージョン 0x030a が定義されています。
__MTEDIT_T	TEXT.H	このマクロが定義された場合、マルチフォントテキストエディットレコードの定義が変更されます。

ここで、マクロ名 __MTEDIT_T の役割について補足しておきます。

SX31KIT では、互換性を保つためマルチフォントテキストエディットレコードを表す構造体 MTEdit を従来のテキストエディットレコード TEdit を拡張する形で宣言しています。しかし、「SX-WINDOW ver.3.1 システムキット」(シャープ) や「シャーペンワープロパック」(計測技研) に含まれているインクルードヘッダ MTEDIT.H では、TEdit とは関係なく、MTedit を宣言しています。SX31KIT ではマクロ名 __MTEDIT_T を定義すると、構造体 MTEdit を後者の形式で宣言します。なお、マクロ名 __MTEDIT_T を使用するときは、必ずマクロ名 SX31KIT をチェックするようにしてください。

ビデオマン関係のマクロ

SX31KITでは、Table A-3に示すようなビデオマン用のマクロをインクルードヘッダVIDEO.Hで定義しています。

Table A-3 ● ビデオマン関係のマクロ

マクロ名	役割
VMSetDuration VMSetTimeScale VMSetRate VMSetCurTime VMSetPlayRect VMSetUserAtom	VMSetParam関数用のマクロ
VMGetDuration VMGetTimeScale VMGetRate VMGetCurTime VMGetUserAtom VMGetWidthHeight VMGetStatus VMGetMediaMode VMGetTotalSample VMGetCurSample VMGetTotalFrame	VMGetParam関数用のマクロ
VGetPlayRect VMGetFrameDuration	VMGetParam2関数用のマクロ

C++での利用

SX31KITは、gcc2[1]などのC++コンパイラでも利用することができます。

C++でSX31KITを利用する場合には、インクルードヘッダを取り込むプリプロセッサ命令#includeの前後をextern "C" {}で囲んで利用するようにしてください。

```
extern "C" {
  ¥#include <sxdef2.h>
};
```

1)Charlie氏が移植したgcc ver.2系統のC/C++/Objective-Cコンパイラ。添付CD-ROMに収録されています。

⊙コラム：アセンブラマクロファイル

SX31KITには、アセンブラで開発するのに必要なマクロファイルも含まれています。アセンブラマクロファイルには、大きく分けて2つの利用目的があります。1つは、アセンブラ言語を用いたプログラム開発で直接利用される場合、もう1つはC言語を用いたプログラム開発の際にCコンパイラから間接的に利用される場合です。SX31KITに付属するアセンブラマクロファイルは、Workroom付属のアセンブラマクロに対する上位互換を保ちつつ、SX-WINDOW ver.3.1で新たに追加された機能をアセンブラから利用できるように設計されています。

SX31KITに付属するアセンブラマクロファイルには以下のものがあります。

○ SXCALL.H

アセンブリ言語を用いてプログラムを作成する際に読み込むマクロファイルです。各種の定数定義や、データ構造の定義を行っています。C言語から使用されることはありません。

○ SXCALL.MAC

SXコール番号、コール名とそのマクロを記述したファイルです。アセンブリ言語を用いてプログラムを作成する際に読み込みます。C言語から使用されることはありません。

○ SXCALL.EQU

gccをSXモードで動かした場合にgccが使用するアセンブラマクロファイルです。正確にはgccが出力するアセンブラファイルからアセンブラによって読み込まれます。SXCALL.EQUでは主にSXコールのコール番号を定義しています。SXCALL.EQUは、内部でDOSCALL.EQUを読み込むようになっています。

SX31KITのインクルードヘッダ内部では、マクロ名__GNUG__が定義されているかどうかをチェックし、定義されていればgcc2であると判断して“typedef enum 列挙型名 型名”の定義を省略します[2)]。Workroomのインクルードファイルをそのまま利用しているCONSOLE.H, CONTROL.H, EVENT.H, RESOURCE.H, SXMEMORY.Hについては、この変更を各自で行ってください。

2)gcc2になってenum型の言語上の定義が変更されたため、そのままではコンパイルエラーになるからです。

LIBSXC便利帳

ここでは、LIBSXC をより高度に活用したり、あるいは LIBSXC を使用していて問題が起こった際の解決の糸口に利用できる情報として、次の項目を取り上げます。

(1) 実行ファイルの構造

LIBSXC を使用した実行ファイルの構造について説明します。

(2) 実行時のメモリマップ

LIBSXC を使用したプログラムの、実行時のメモリの状態について説明します。

(3) スタートアップのプロセス

LIBSXC を使用したプログラムを起動したときに、ライブラリがどのような準備をするかについて説明します。

(4) 主なライブラリ内部変数

LIBSXC を使用した場合に、プログラム中から参照できるライブラリ内部変数について説明します。

ただし、これらの内部情報を利用すると、LIBSXC ライブラリのバージョンや、特定の動作環境に依存することになり、互換性が損われる可能性があるので注意してください。

実行ファイルの構造

Fig.B-1 に LIBSXC をリンクした場合の実行ファイルの構造を示します。

Fig. B-1 ● LIBSXC をリンクした場合の実行ファイルの構造

(a)実行ファイルヘッダ
(b)textセクションの内容
(c)セクション情報
(d)rdataセクションの内容
(e)rldataセクションの内容
(f)相対オフセットテーブル
(g)dataセクションの内容

(a) から (g) の各セクションの内容は次のとおりです。

(a) 実行ファイルヘッダ

Human68k のコマンドラインから起動するために必要なヘッダです。SX–WINDOW のモジュールヘッダではありません。

(b) `text` セクションの内容

プログラム本体です。SX–WINDOW のモジュールヘッダはこの部分に含まれます。

(c) セクション情報

リンカ (`hlk`) が出力するセクション情報です。相対セクションを含む、各セクションのサイズが記述されています。

(d) `rdata` セクションの内容

オフセットが 64K バイト以内の初期化済みデータセクションの内容です。

(e) `rldata` セクションの内容

オフセットが 64K バイトを超える初期化済みデータセクションの内容です。

(f) 相対オフセット (`roffset`) テーブル

相対セクションを初期化する際、アドレスに依存する部分を補正す

るためのテーブルです。

(g) data セクションの内容

通常の初期化済みデータセクションの内容です。

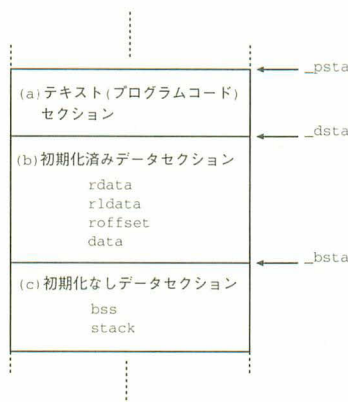
(a)、(b) および (g) は通常の実行ファイルと同じです。(c) から (f) は、LIBSXC 独自 (正確には X680x0 gcc 独自) の部分で、rdata をはじめとする相対セクションのためにあります。相対セクションに関する詳細は、コラム「セクション」(p.23) を参照してください。

B・3 実行時のメモリマップ

タスク間で共有される領域

Fig. B-2 に、LIBSXC をリンクしたプログラムを実行した場合のタスク間で共有される領域のメモリマップ[1]を示します。

1) プログラム実行時の、メモリの状態図のこと。

Fig. B-2 ● LIBSXC をリンクした場合のプログラム実行時に、タスク間で共有される領域

(a) から (c) の各セクションの内容は次のとおりです。

(a) テキスト (プログラムコード) セクション

プログラム本体が格納されています。この領域の先頭アドレスは、ライブラリ内部変数 _psta が指しています。

(b) 初期化済みデータセクション

あらかじめ内容がわかっている共用大域変数[2]が格納されています。この領域の先頭アドレスはライブラリ内部変数 _dsta が指しています。

2) たとえば、"common int a = 1;" といったもの。

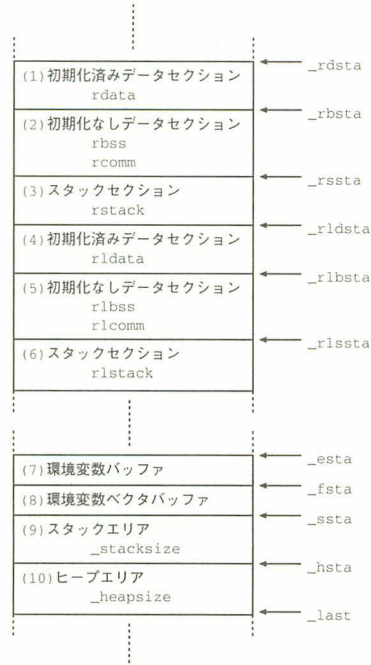
(c) 初期化なしデータセクション

あらかじめ内容がわかっていない共用大域変数[3]が格納されています。この領域の先頭アドレスはライブラリ内部変数 _bsta が指しています。

3) たとえば "common int a;" といったもの。

タスクごとに確保される領域

Fig. B-3 に、LIBSXC をリンクしたプログラムを実行した場合のタスクごとに確保される領域を示します。

Fig. B-3 ● LIBSXC をリンクした場合のプログラム実行時に、タスクごとに確保される領域

(1) から (10) の各領域の内容は次のとおりです。

(1) 初期化済みデータセクション

あらかじめ内容がわかっているタスクごとの大域変数[4]が格納されています。 この領域の先頭アドレスはライブラリ内部変数 _rdsta が指しています。

4)たとえば、“int a = 1;” といったもの。

(2) 初期化なしデータセクション

あらかじめ内容がわかっていないタスクごとの大域変数[5]が格納されています。この領域の先頭アドレスはライブラリ内部変数 _rbsta が指しています。

5)たとえば、“int a;” といったもの。

(3) スタックセクション

この領域は LIBSXC では使用されません。領域のサイズも 0 です。

実行時のスタックとしては (9) の領域のみが使用されます。この領域の先頭アドレスはライブラリ内部変数 _rssta が指しています。

(4) 初期化済みデータセクション

あらかじめ内容がわかっているタスクごとの大域変数[6]が格納されています。この領域の先頭アドレスはライブラリ内部変数 _rldsta が指しています。

6)たとえば、"remote int a = 1;" といったもの。

(5) 初期化なしデータセクション

あらかじめ内容がわかっていないタスクごとの大域変数[7]が格納されています。この領域の先頭アドレスはライブラリ内部変数 _rlbsta が指しています。

7)たとえば、"remote int a;" といったもの。

(6) スタックセクション

この領域は LIBSXC では使用されません。領域のサイズも 0 です。実行時のスタックとしては、(9) の領域のみが使用されます。この領域の先頭アドレスはライブラリ内部変数 _rlssta が指しています。

(7) 環境変数バッファ

タスクが起動したときに親タスクが持っていた環境変数の情報が、引数ごとに NULL キャラクタで区切られて格納されています。main 関数の第 3 引数の各要素の実体でもあります。この領域の先頭アドレスはライブラリ内部変数_esta が指しています。

(8) 環境変数ベクタバッファ

(7) の環境変数バッファに格納された環境変数の各要素へのポインタが、配列として格納されています。main 関数の第 3 引数の各要素でもあります。この領域の先頭アドレスはライブラリ内部変数 _fsta が指しています。

(9) スタックエリア

タスクが実行時に使用するスタックとして使用される領域です。この領域の先頭アドレスはライブラリ内部変数 _ssta が指しています。MC680x0 では、スタックは下位アドレスに向かって消費されるので、実際にはこのスタックセクションの最後から順次使用されます。スタックポインタが、スタックエリアの先頭を越えてしまった状態を「スタックオーバーフロー」と呼びます。この場合、(2) や (1) の領域が破壊されることになり、さらにオーバーフローが進むと他のタスクの領域までも破壊していきます。

スタックセクションのサイズは、ヒープ領域と違って、足りなくなったからといって動的に増やすことはできませんので、スタックをたくさん消費するようなプログラムでは、あらかじめスタックを多めに確保するようにしてください。なお、デフォルトのスタックのサイズは 32K バイトです。

(10) ヒープエリア

この領域は LIBSXC では使用されません。領域のサイズも 0 です。この領域の先頭アドレスはライブラリ内部変数 _hsta が指しています。LIBSXC のヒープ領域については、「メモリの動的確保」 (p.150) を参照してください。

スタートアップのプロセス

プログラムが起動してから main 関数が実行されるまでに、LIBSXC のスタートアップルーチンが行う動作について順に説明します。

(1) カーネルの起動

Human68k のコマンドラインから起動された場合、ここから実行されます。まず、指定のカーネルを起動します。カーネル起動後、モジュールヘッダに記されたアドレスに飛びます。SX シェル上で起動した場合、カーネルを起動する必要はありませんから、次の「(2) ワークエリアの確保」から実行されます。

(2) ワークエリアの確保

セクション情報をもとにタスクごとに必要なワークエリアのサイズを計算し、再配置不可能なブロックを 1 つ確保します。確保されたブロックへのハンドルは、ライブラリ内部変数 _data_hdl1 に格納されます。さらに確保されたワークエリアの先頭アドレスに 0x8000 を足した値が A5 レジスタに代入されます。これはタスクごとの変数が、A5 レジスタ相対アドレッシングでアクセスされるためです。

(3) ワークエリアの初期化

次に「(2) ワークエリアの確保」で確保されたタスクごとのワークエリアを初期化します。タスク間で共有領域にある rdata, rldata セクションの内容を、タスクごとのワークエリアにコピーします。また、rbss, rlbss 領域をクリアします。

(4) コマンドラインの解析

コマンドライン文字列を解析し、main 関数に渡すための引数ベクタ[1]を作成します。これらの領域は rbss 中に置かれます。

1)main(int argc, char* argv[], char* envp[]) の argc と argv[]

(5) 環境変数およびスタックエリアの確保

環境変数およびスタックエリアに必要なサイズを計算し、再配置不可能なブロックを 1 つ確保します。環境変数とスタックエリアは、ともにこの再配置不可能なブロックのなかに置かれます。確保

されたブロックへのハンドルは、ライブラリ内部変数_data_hdl2に格納されます。

次に、main 関数に渡すために、環境変数をコピーし、環境変数ベクタ[2]を作成します。さらに、実行時のスタックとして SP(A7) レジスタを設定します。

2)main(int argc, char* argv[], char* envp[]) の envp[]

(6) main 関数の実行

その他必要なライブラリ内部モジュールや内部変数を初期化したあと、main 関数が実行されます。

(7) 後処理

main 関数終了後、オープンしたままのファイルをクローズしたり、ヒープ領域を開放するなどの後処理をしたあと、「(2) ワークエリアの確保」や「(5) 環境変数エリアおよびスタックエリアの確保」で確保したハンドルを開放し、TSExit をコールして終了します。

そのほか、LIBSXC のスタートアップルーチンに関する詳細は、LIBSXC ソースコードパッケージ中の以下のソースファイルを参照してください。

○ src/sxstartup/_start.c

主なライブラリ内部変数

1)ソフトバンク刊『X680x0 libc』『X680x0 Develop & libc II』を参照してください。

LIBSXC のライブラリ内部変数は、LIBC ですでに公開されているもの[1]に関しては、LIBSXC でもその役割や内容はほぼ同じです。したがって、ここでは LIBC とは働きが違うものや、LIBSXC 独自のものについて説明します。

○ OBJECTTYPE
モジュールヘッダ内のモジュールタイプを指定します。デフォルトでは “OBJR” になっています。

○ _COMMON_SIZE
モジュールヘッダ内のコモンエリアのサイズを指定します。gcc の SX モードにおけるコモンセクションとは異なりますので注意してください。デフォルトのサイズは 0 です。

○ extern int _common;
SX-WINDOW によって確保されたコモンエリアへのポインタです。プログラム起動時に、SX-WINDOW によって自動的に設定される A4 レジスタの値が代入されています。

○ extern common char _sxkernelcomm[];
Human68k のコマンドラインから起動された場合に使用するカーネルを指定します。デフォルトでは “sxkernel” です。詳細は「カーネルの変更」(p.149) を参照してください。

○ struct _mep _memcp; / struct _psp _procp;
それぞれカーネルのプロセスのメモリ管理ポインタ、プロセス管理ポインタを指しています。つまり、sxwin.x や sxkernel.x に関しての情報であって、タスクそれ自身の情報ではありません。

○ extern long _pid;
自分自身のタスク ID です。プログラム起動時に SX-WINDOW によって自動的に設定される D0 レジスタの値が代入されています。

○ extern long _ppid;

コードを共有するタスクの ID です。プログラム起動時に、SX-WINDOW によって自動的に設定される D1 レジスタの値が代入されています。

○ `extern void* _rdsta; / extern void* _rbsta; / extern void* _rssta; / extern void* _rldsta; / extern void* _rlbsta; / extern void* _rlssta;`
タスクごとに確保された各相対セクションの先頭アドレスです。

○ `extern void* _data_hdl1;`
タスクごとに確保されるワークエリアへのハンドルです。

○ `extern void* _data_hdl2;`
タスクごとに確保される環境変数およびスタックエリアへのハンドルです。

○ `extern common sizeinfo_t __size_info;`
リンカ (hlk) が出力する相対セクション情報です。sizeinfo_t 構造体は次のとおりで、それぞれのメンバは各セクションのサイズを示しています。

```
typedef struct {
    size_t text;
    size_t data;
    size_t bss;
    size_t comm;
    size_t stack;
    size_t rdata;
    size_t rbss;
    size_t rcomm;
    size_t rstack;
    size_t rldata;
    size_t rlbss;
    size_t rlcomm;
    size_t rlstack;
    size_t roffset;
    size_t reserved[2];
} sizeinfo_t;
```

以上のライブラリ内部変数は、プログラム起動時に初期化され、実行

中にその内容が変わることはありません。これらの変数はライブラリの実行に深く関わっているので、不用意に書き換えた場合の動作は保証できません。

これらのライブラリ内部変数の利用は参照のみにとどめてください。

SX-WINDOW 対応フリーソフト一覧

これまで、多くのフリーソフトウェア作者の手によって、さまざまな力作が SX–WINDOW のために、あるいは SX–WINDOW をプラットフォームにして開発されてきました。本節では、これらのフリーソフトウェアの一覧を示します。なお、ここに示したフリーソフトウェアは、本書に添付の CD–ROM に収録されています。

No.1	3D グラフ.X	分類	¥APL¥3DGRAPH
作者名	高橋博樹 (HI–Br.)		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	SXCalc.X (カップめん氏作) や CGraph.X (高橋博樹 (HI–Br.) 氏作) のデータをもとにして 3D 棒グラフや折れ線グラフを描画するプログラム。		

No.2	CALENDAR.X	分類	¥APL¥CALEN
作者名	MAPI(小野塚 正則)		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	12、16、24 ドットの 3 種類からフォントを選択したり、祝日や振り替え休日を赤で表示したり、休日データを任意に設定したりすることができるプログラム。		

No.3	cddata.x	分類	¥APL¥CDDATA
作者名	とーい		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	SX–WINDOW 上で CD(コンパクトディスク) の曲目管理とカセットテープ、MD(ミニディスク)、DAT などのラベル作成支援を行うプログラム。		

No.4	CGraph.X	分類	¥APL¥CGRAPH
作者名	高橋博樹 (HI–Br.)		
対応 OS	SX 3.1	収録内容	別途ドキュメントアーカイブあり。別途ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	SX–WINDOW 上で動作するグラフ描画&近似式計算ツール。		
備考	グラフメニュー.X (高橋博樹 (HI–Br.) 氏作)、SXCalc.x (カップめん氏作) 対応、SX-BASIC 対応。		

No.5	データ間引.X	分類	¥APL¥DATAUTY1
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SXCalc.X (カップめん氏作) や CGraph.X (高橋博樹 (HI-Br.) 氏作) のデータを適宜削除するユーティリティプログラム。		

No.6	非線形近似.X	分類	¥APL¥DATAUTY2
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SXCalc.X (カップめん氏作) や CGraph.X (高橋博樹 (HI-Br.) 氏作) に入力されたデータから、「$Y = a\,sin\,X + b\,cos\,X + \ldots$」 のような関数で近似値を作成するためのユーティリティプログラム。		

No.7	SXdentaku.x	分類	¥APL¥DENTAKU
作者名	BAA		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で動作する、少し高機能な電卓プログラム。		

No.8	グラフメニュー.X	分類	¥APL¥GMENU
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	別途グラフモジュールアーカイブあり。別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SXCalc.X (カップめん氏作) や CGraph.X (高橋博樹 (HI-Br.) 氏作) などのコピーデータを利用してグラフを描画するためのプログラム。		
備考	SX-BASIC 対応。		

No.9	SX-gnuplot.x	分類	¥APL¥GNUPLOT
作者名	MAPI (小野塚 正則)		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	グラフ作成ソフト GNUPLOT の SX-WINDOW 版。PAT4、PIX 形式でのセーブ、スクラップへの転送が可能。		

No.10	面グラフ.X、3D 円グラフ.X	分類	¥APL¥GRAPH
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SXCalc.X (カップめん氏作) や CGraph.X (高橋博樹 (HI-Br.) 氏作) などのコピーデータを利用して面グラフ、3D 円グラフを描画するためのプログラム。		
備考	グラフメニュー.X (高橋博樹 (HI-Br.) 氏作) 必須ソフト。		

No.11	3D 棒グラフ.X、横棒グラフ.X、レーダー型グラフ.X	分類	¥APL¥GRAPH2
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SXCalc.X (カップめん氏作) や CGraph.X (高橋博樹 (HI-Br.) 氏作) などのコピーデータを利用して 3D 棒グラフ、横棒グラフ、レーダー型グラフを描画するためのプログラム。		
備考	グラフメニュー.X (高橋博樹 (HI-Br.) 氏作) 必須ソフト。		

No.12	MCZector.x	分類	¥APL¥MCZ
作者名	仁泉大輔		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で動作する、ベクトルフォント (Zeit 社のフォントなど) のマージ兼クリーナ。		

No.13	メモリメーター.x	分類	¥APL¥MEMG
作者名	VSYNC		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	全メモリ容量、使用可能メモリ容量、連続使用可能メモリ容量をグラフィカルに表示したり、メモリブロックの配置を最適化したりするプログラム。		

No.14	mvsi.x	分類	¥APL¥MVSI
作者名	けんと		
対応 OS	SX 3.1	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW システムアイコンの一部をアニメーションさせるプログラム。		

No.15	postSx.x	分類	¥APL¥POSTSX
作者名	仁泉大輔		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	住所をパラメータとして郵便番号を検索・表示するプログラム。		

No.16	55ED.X	分類	¥APL¥SC55ED
作者名	しゅうさく		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で、Roland SC-55mkII のパッチパラメータをエディットするためのプログラム。		
備考	ZMUSIC.X ver.2.00 以上、または RCD.X ver3.01 以上が必要。		

No.17	SXCalc.x	分類	¥APL¥SXCALC
作者名	カップめん		
対応 OS	SX 3.0 以上	収録内容	別途詳細ドキュメントアーカイブあり。別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ウィンドウの四分割や簡単なコマンドマクロを実装した、SX-WINDOW 上の EXCEL 風の表計算プログラム。		
備考	SYLK 形式、CSV 形式、タブテキスト形式の読み込みが可能。シェアウェア (6,800 円)。		

No.18	DBCtr.x	分類	¥APL¥SXDB
作者名	けい太		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で動作する dBASE 型データベース。		

No.19	sxmemo.x	分類	¥APL¥SXMEMO
作者名	GRANADA		
対応 OS	SX 1.1 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上でポストイット感覚で使えるノートアクセサリ。		

No.20	負荷計測 SX.X	分類	¥APL¥SXPM
作者名	VSYNC		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	負荷計測 SX.X が起動された時点での MPU 負荷 (1 秒間に何回アイドルイベントが通知されるか) を計測し、それをワークステーション風に表示するプログラム。		

No.21	小時計.r/極小時計.r/トケイ.r/CLK.r/CK.r	分類	¥APL¥SXTOKEI
作者名	PRAY		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	メモリ使用量の少ない (0.6K バイト 〜 1K バイト強) 各種の時計の詰め合わせセット。		

No.22	とけい.x	分類	¥APL¥TOKEI
作者名	Zeek		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	30 分ごとの時報、アラーム機能、指定された日時に指定プログラムの制御が行えるなど、高機能な鳩時計プログラム。		

No.23	xtarante.x	分類	¥APL¥XTRANTE
作者名	T·A·N, Natchsoft(なっち)		
対応 OS	SX 3.1	収録内容	別途オンラインヘルプアーカイブあり
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で動作する Excel ライクな表計算ソフト。		
備考	メインメモリ 4M バイト以上必要。		

No.24	Zeitor.x	分類	¥APL¥ZEITOR
作者名	仁泉大輔		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で動作する、ベクトルフォントの CAD プログラム。		

No.25	sxgzip.x	分類	¥ARC¥SXGZIP
作者名	BAA		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ファイル圧縮プログラム GNU zip ver.1.2.4 の SX-WINDOW 版。コマンドラインモード、ウィンドウモード、サーバモードの 3 種類があるプログラム。		
備考	ViSON.x (Niggle 氏作)、SXXX.X (解凍 SX：もけもけ氏作) で使っているクライアントーサーバ型のインターフェースをサポート。		

No.26	sxtar.x	分類	¥ARC¥SXTAR
作者名	Niggle		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	解凍 SX (もけもけ氏作) のインターフェースを使った解凍専用の TapeArchiver。		
備考	サーバモードしかないため、ViSON.x (Niggle 氏作) と組み合わせて使用しなければならない。		

No.27	SXXX.X	分類	¥ARC¥SXXX
作者名	もけもけ		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	高圧縮書庫管理プログラム LHa.X ((.EXE) 吉崎栄泰氏作) で圧縮されたファイル (lhd, lh0, lh1, lh4, lh5, lh6 形式) を SX-WINDOW 上で解凍するプログラム。		

No.28	ViSON.x	分類	¥ARC¥VISON
作者名	Niggle		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX シェル上でアーカイブ中のファイルを参照することができるプログラム。di.r (SX-WINDOW の内部コマンド) の操作性を継承、拡張したもの。		
備考	LZH ファイルと tar ファイル、gzip ファイルに対応可能。		

No.29	トランプ.LB	分類	¥DATA¥CARD
作者名	ASO		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で使うことができるトランプカードフリーリソースデータ。トランプカードデータを使ったアプリケーション開発時の手間を省くことができる。		

No.30	麻雀牌 mini.LB	分類	¥DATA¥PAIMIN
作者名	結城あすか		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で使える麻雀牌フリーリソースデータ 麻雀牌.LB の縮小版。		

No.31	麻雀牌.LB	分類	¥DATA¥PAISTD
作者名	結城あすか		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で使える麻雀牌フリーリソースデータ。麻雀牌を使ったアプリケーション開発時の牌データ作成の手間を省くことができる。		

No.32	SX_logo.pan	分類	¥DATA¥SXLOGO
作者名	Ryu!		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SxPANIC.r (Sho-Ta 氏作) のための panic データのスクリーンセーバサンプル。		
備考	SxPANIC.r (Sho-Ta 氏作)、SAdjust.r (Sho-Ta 氏作) が必要。		

No.33	UNO.LB	分類	¥DATA¥UNO
作者名	B-Head		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で使えるカードゲーム「UNO」用フリーリソースデータ。「UNO」のカードデータを使ったアプリケーション開発時の手間を省くことができる。		

No.34	目玉.x	分類	¥DESKTOP¥EYE
作者名	はばしん		
対応 OS	SX 2.01 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	目玉とサイズボックス以外の部分が透けて見える、オリジナルの WDEF リソースを用意した、いわずと知れた xeyes (目玉) もどき。		

No.35	MISA.X	分類	¥DESKTOP¥MISA
作者名	Ussy		
対応 OS	SX 1.1 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	NIFTY-Serve の FGALAV に登録されているアニメーション画像データ「未沙ちゃん」シリーズの MS-Windows 3.0 版「美砂ちゃん」を移植したもの。画面の隅においておけば幸せになれるかも。		

No.36	SeSS.x	分類	¥DESKTOP¥SESS
作者名	特α級電脳師		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW のアプリケーションの起動・終了をコントロールしてスクリーンセーバのようにみせるプログラム。		

No.37	X-logo.SMD, Life.SMD, Swarm.SMD	分類	¥DESKTOP¥SXMODULE
作者名	An		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW デスクアクセサリ集対応のスクリーンセーバモジュール 3 点。		

No.38	arlk.x	分類	¥DEVELOP¥ARLK
作者名	けんと		
対応 OS	SX 3.1 / Human68k	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX シェル上から呼び出すことが可能な高機能リソースリンカプログラム。シャープ純正 rlk.x とは非互換。		

No.39	cccv.x	分類	¥DEVELOP¥CCCV
作者名	BLACK.		
対応 OS	Human68k	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	CCV.X により変換されたコードリソースを元の .X ファイルに戻すツール。		
備考	シンボル情報については完全に復元することはできない。		

No.40	cplk.x	分類	¥DEVELOP¥CPLK
作者名	Dangerous Driver ZNT		
対応 OS	Human68k	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 対応の動画ファイル (CGA 形式) の ADPCM データを追加、交換、抽出するプログラム。		

No.41	lvcv.x, lvcv030.x	分類	¥DEVELOP¥LVCV
作者名	Dangerous Driver ZNT		
対応 OS	Human68k	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	CZ-6VS1 (カラーイメージユニット 2) 付属のライブスキャン.X (SX-WINDOW 上で動作) が出力する .LV 形式の動画ファイルを、SX-WINDOW の .GLM 形式でコマ別に出力するプログラム。CGA データの作成にも使える。		
備考	gcc の-m68040 オプション付きで作成された X68030 以上専用の lvcv030.x を同梱。		

No.42	MenuDesigner.x	分類	¥DEVELOP¥MENUDESI
作者名	仁泉大輔		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW のメニューリソース MENU を編集するためのプログラム。		

No.43	rscv.x	分類	¥DEVELOP¥RSCV
作者名	JunT		
対応 OS	Human68k	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW で用いられるリソースの内容を 16 進ダンプしたり、PAT4 などのパターンリソースならば画面に表示することができるプログラム。		

No.44	SXREF.DIF, SXREF.TXT	分類	¥DOC¥INSIDE
作者名	亜山　雪		
対応 OS	SX 2.0 以上	収録内容	差分のみ
サポートネット	NIFTY-Serve		
紹介	インサイド SX.X (開発ドキュメント用ツール集に同梱) に対応した、SX アプリを開発するのに便利なオンラインドキュメント。		
備考	インサイド SX.X が必要。		

No.45	VMINSTALL.PEN	分類	¥DOC¥VMINST
作者名	BLACK.		
対応 OS	SX 3.1	収録内容	ドキュメント内にサンプルソースファイルあり
サポートネット	Network-SX NG		
紹介	IVM 用のビデオマネージャリソース (VMZAU, VMCUT など) のインストールスクリプトに関するサンプルドキュメント。		
備考	コンパイルするためには開発キットツール集と XC が必要。		

No.46	マップエディタ.X	分類	¥EDITOR¥MAPEDIT
作者名	CHIHI		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で動作する BG マップエディタ。		

No.47	xemacs.x(mule)	分類	¥EDITOR¥MULE
作者名	BAA		
対応 OS	SX 3.1 / Human68k	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	高機能・カスタマイズ可能なエディタ Mule-1.1.04 の SX-WINDOW / Human68k 共用版。		

No.48	nemacs.x	分類	¥EDITOR¥NEMACS
作者名	沖@沖		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	高機能・カスタマイズ可能なエディタである GNU Emacs の日本語版 Nemacs 3.3.2 の SX-WINDOW 版。		
備考	1M バイト以上の空きメモリが必要。		

No.49	ng.x	分類	¥EDITOR¥SXNG
作者名	沖@沖		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX-WINDOW 上で動作する日本語 micro GNU Emacs、Ng 1.3.1 。		

No.50	2 行にするの.r	分類	¥EXT¥2LINE
作者名	けんと		
対応 OS	SX 3.01/3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	長いファイル名を 2 行に折り返して表示することができるプログラム。		
備考	adi.r、アイコン間隔.r (けんと氏作) と組み合わせると最高。		

No.51	ActiveJmp.X	分類	¥EXT¥ACTJP
作者名	PRAY		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	アクティブになったウィンドウが画面内に収まっていない、あるいは画面内にない場合、ウィンドウ全面が収まるよう最小限のスクロールを自動的に行うプログラム。		
備考	Winselect.x (T·A·N 氏作) と組み合わせると最高。		

No.52	adi.r	分類	¥EXT¥ADI
作者名	けんと		
対応 OS	SX 3.01/3.1 のみ	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	1) アクティブでないときもポップアップメニューを出す、2) ウィンドウを開いたときにアイコンを整頓する、3) 沖@沖氏の lndrv のディレクトリリンクに対応、4)DIRDTOP.SX を作るパスを指定するなど、11 項目にわたり di.r を拡張したシステムプログラム。		
備考	アイコン間隔.r、2 行にするの.r (いずれも けんと氏作) と組み合わせると最高。BUILTIN.LB に組み込むインストールパッチプログラム付き。		

No.53	BeepChanger.x	分類	¥EXT¥BEEPCHG
作者名	MAPI(小野塚 正則)		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW の「ピンポーン」という BEEP 音を変更するためのプログラム。		

No.54	cg.bfd	分類	¥EXT¥CGBFD
作者名	沖@沖		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	キャンバス.X と CG ビジョン.X のメニューを ClickMenu (沖@沖氏作) と同様、マウスのクリックで ON/OFF できるようにするためのプログラム。		

No.55	ClickMenu.x	分類	¥EXT¥CLICK
作者名	沖@沖		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ポップアップメニューをマウスのクリックで ON/OFF できるようにするためのプログラム。		
備考	階層化メニュー.x に同等の機能あり。		

No.56	地上げ屋.x	分類	¥EXT¥DIAGE
作者名	もけもけ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	「プログラミング専用」「音楽観賞専用」のように、5 つのデスクトップを設定することができるプログラム。		

No.57	e-c-brace.ex/isearch.ex/optab.ex/rpar.ex/setkind1.ex/xclick.ex	分類	¥EXT¥EXC
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	インクリメンタルサーチ等を可能にする、シャーペン.X の外部コマンド。		
備考	SX-WINDOW ver.3.1 付属のシャーペン.X が必要。詳しくは「Oh!X」1994 年 10 月号を参照。		

No.58	extdrag.r	分類	¥EXT¥EXTDRAG
作者名	KUM		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	extdrag.r (Niggle 氏作 ViSON に同梱) のソースファイルを改造した、GARBAGE.X と併用時の不具合を取り除くプログラム。		

No.59	ファイル名補完.X	分類	¥EXT¥FCMPL
作者名	Yoz.		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX-WINDOW のテキストエディット上 (ファイル名を入力する疑似ダイアログ等) でファイル名の補完を可能にするプログラム。		

No.60	fix_mv.r	分類	¥EXT¥FIXMV
作者名	けんと		
対応OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	subst.x や mount.x で複数のドライブを単一ドライブにまとめたときの、SX-WINDOW 上での不都合を解消するプログラム。		

No.61	HCDAD.SYS	分類	¥EXT¥HCDAD
作者名	PRAY		
対応OS	Human68k	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	連奏タイプの CD-ROM ユニットの指定 LUN 番号を無視することにより、CD-ROM ユニットの起動時間を短縮するプログラム。		
備考	cddev.sys (計測技研製 CD-ROM デバイスドライバ) より先に登録する必要がある。		

No.62	henwin2.x	分類	¥EXT¥HENWIN2
作者名	Niggle		
対応OS	SX 3.1	収録内容	NeXTlike-Window 用のリソース同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ASK (FEP) の変換ウィンドウの位置、サイズ、フォントサイズ (12/16 ドット) を変更することができるプログラム。		
備考	ASK 専用高速化モードを使用すると不具合あり。		

No.63	アイコン間隔.r	分類	¥EXT¥IWIDTH
作者名	けんと		
対応OS	SX 3.01 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	di.r などでアイコンが並べられる間隔 (縦横 80 ドット) を任意の値に調整するプログラム。		
備考	adi.r、2行にするの.r (けんと氏作) と組み合わせると最高。		

No.64	階層化メニュー.X	分類	¥EXT¥KAISO
作者名	CHIHI		
対応OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX-WINDOW のメニューマネージャを、シェルメニューの階層化、表示文字列のマルチフォント化、PAT4 イメージの貼り付けなど、8項目にわたって機能強化するプログラム。		

No.65	SXperiod.x	分類	¥EXT¥MPERI
作者名	あーるもーりあ		
対応OS	SX 3.1 / Human68k	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW をマルチピリオドに対応させるための常駐プログラム。		

No.66	SAdjust.r	分類	¥EXT¥SADJUST
作者名	Sho-Ta		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ADJUST.r (SX-WINDOW 内部コマンド) を使わずに、SX-WINDOW の実画面を任意サイズに設定するプログラム。		
備考	CRTCTRL.r (Sho-Ta 氏作) といっしょに利用することにより、マウスオペレーションでセットアップ可能。SX-BASIC 対応。		

No.67	setdtop.x	分類	¥EXT¥SETDTOP
作者名	Mew		
対応 OS	SX 2.01 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	DIRDTOP.SX を指定パスにまとめるプログラム。		

No.68	SilentActivate.X	分類	¥EXT¥SILENT
作者名	CHIHI		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	奥に表示されたウィンドウを手前に並べ換えることなく、アクティベートにするためのプログラム。		
備考	SX-BASIC 対応。専用インストーラ付属。コントロールパネル追加設定可能。		

No.69	SNAP04.INS, SNAP04.LB	分類	¥EXT¥SNAP
作者名	An		
対応 OS	SX 3.1	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上でダイアログを利用するときに、マウスカーソルを「設定」または「確認」などのボタンに自動的に移動させるプログラムのインストールスクリプト。		
備考	インストーラ.X が必要。コントロールパネルに追加設定可能。		

No.70	SXerror.x	分類	¥EXT¥SXERROR
作者名	沖@沖		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX-WINDOW 上でバスエラーやアドレスエラーが発生したタスクの ID を表示し、そのタスクのみを強制終了することができるようにするプログラム。		

No.71	SXWS.x	分類	¥EXT¥SXWS
作者名	カップめん		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	デスクトップ上でつねに最も手前に表示される、SX-WINDOW 標準機能のページアイコンと同じような動作をするプログラム。		

No.72	WL203.INS, WL203.LB	分類	¥EXT¥WINLOC
作者名	An		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で新しく開くウィンドウの位置を細かく制御するためのインストールスクリプト。		
備考	インストーラ.X が必要。コントロールパネルに追加設定可能。		

No.73	WinSelect.x	分類	¥EXT¥WINSEL
作者名	T·A·N		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	キーボードからアクティブウィンドウの切り替えやディレクトリ表示ウィンドウのスクロール、ヒストリ、タスクの終了などを実行するためのプログラム。		
備考	ActiveJmp.x (PRAY 氏作) と組み合わせることにより、マウスレスオペレーションが可能。		

No.74	WorkSpace.x	分類	¥EXT¥WORKSPC
作者名	T·A·N		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	6 個の仮想デスクトップ (ワークスペース) を提供するプログラム。		
備考	管理できるウィンドウは最大 64 まで。		

No.75	disk_info.x	分類	¥FILE¥DINFO
作者名	ウィル		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	高密度な情報表示を行うことができる、dinfo.r (SX-WINDOW の内部コマンド) の上位互換プログラム。		

No.76	DRVINFO.R	分類	¥FILE¥DRVINFO
作者名	VSYNC		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	指定ドライブの容量変化をリアルタイム表示したり、容量使用率を円グラフで表示するなど、高密度な情報表示を行う dinfo.r (SX-WINDOW の内部コマンド) の上位互換プログラム。		
備考	BUILTIN.LB に登録する方法も記載。		

No.77	FILEINFO.X	分類	¥FILE¥FILEINFO
作者名	VSYNC		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ファイルのパスをスクラップへコピーしたり、複数ファイルのファイル属性をまとめて変更したり、複数ファイルの容量表示をしたりするなど、高密度な情報表示を行う info.r (SX-WINDOW の内部コマンド) の上位互換プログラム。		
備考	BUILTIN.LB に登録する方法も記載。		

No.78	file_info.x	分類	¥FILE¥FINFO
作者名	ウィル		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	ファイルの実行属性、リンク属性の表示・設定や、フルパス名のクリップボード転送などを行うことができる、info.r (SX-WINDOW の内部コマンド) の上位互換プログラム。		

No.79	fmemo.x	分類	¥FILE¥FMEMO
作者名	ウィル		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	file_info.x (ウィル氏作) と組み合わせることにより、ファイルごとにメモ書き機能を持たせることができるプログラム。		

No.80	SXINFO.R	分類	¥FILE¥SXINFO
作者名	VSYNC		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	現在の SX-WINDOW の空きメモリ容量や 1 秒あたりにタスクに通知されるアイドルイベント数、現在年月日、現在時刻を表示したり、メモリの再配置を行うシステム表示プログラム。		

No.81	gnuchess.x	分類	¥GAME¥GNUCHESS
作者名	Yoz.		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	人間とコンピュータが対戦することも、コンピュータ同士で対戦することもできるチェスゲーム GNU CHESS の SX-WINDOW 版。		

No.82	MKCS.X	分類	¥GAME¥GOLFEDIT
作者名	T.KAZBON		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SxGOLF.X (T.KAZBON 氏作) のコースデータを作成するためのプログラム。		
備考	16 色モード時のみ動作可能。		

No.83	GOLFSW.PCM, CASTA.PCM, CUP_IN.PCM, SPRING.PCM, SPLASH.PCM, APL.PCM	分類	¥GAME¥GOLFPCM
作者名	T.KAZBON		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SxGOLF.X (T.KAZBON 氏作) 用のロイヤリティフリーの ADPCM データ。		
備考	リソースリンカ等が必要。		

No.84	SX-PITMAN.X	分類	¥GAME¥PITMAN
作者名	MAPI (小野塚 正則)		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	電脳倶楽部 Vol.16 に掲載された PITMAN の SX–WINDOW 版。		

No.85	SX-Sirtet.x	分類	¥GAME¥SIRTET
作者名	MAPI (小野塚 正則)		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	SX–WINDOW 上で動作する、4 種類のタイルからなるブロックを横 1 列に並べて同種のタイルを消していく「テトリス」ライクなゲーム。		

No.86	SX 青海.X	分類	¥GAME¥SXCHN
作者名	結城あすか		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	NIFTY–Serve		
紹介	麻雀牌を使ったパズルゲーム。		
備考	麻雀牌フリーリソースデータ麻雀牌.LB、麻雀牌 mini.LB (いずれも結城あすか氏作) のいずれかが必要。		

No.87	SxGOLF.X	分類	¥GAME¥SXGOLF
作者名	T.KAZBON		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	SX–WINDOW 上のゴルフゲーム。4 人まで参加でき、マッチプレイモード (vs. X68000) を楽しんだり、スライスやバックスピンをかけたりすることが可能。		
備考	SX–WINDOW ver.3.0 以上では 16 色モード時のみ動作可能。		

No.88	SX 香港.X	分類	¥GAME¥SXHONG
作者名	うめ吉		
対応 OS	SX 2.01 以上	収録内容	バイナリのみ
サポートネット	NIFTY–Serve		
紹介	麻雀牌を使ったパズルゲーム。		
備考	麻雀牌フリーリソースデータ麻雀牌.LB、麻雀牌 mini.LB (いずれも結城あすか氏作) のいずれかが必要。		

No.89	SXBOMB.x/SX2DMaze.x/SX3DMaze.x	分類	¥GAME¥SXJUNKS
作者名	yana		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY–Serve		
紹介	爆弾を探すパズルゲーム (SXBOMB.X)、2D 迷路のオートデモ (SX2DMaze.x)、3D ダンジョン風迷路 (SX3DMaze.x) の 3 つのプログラム。		

No.90	SX クロンダイク.x	分類	¥GAME¥SXKRON
作者名	Dust		
対応 OS	SX 2.01 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	MS-Windows ver.3 付属のカードゲーム「ソリティア」の SX-WINDOW 版。		
備考	トランプカードフリーリソースデータ、トランプ.LB (ASO 氏作) が必要。		

No.91	マインスイーパ.X	分類	¥GAME¥SXMIN
作者名	An		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	MS-Windows 等でおなじみの隠された爆弾を探すパズルゲーム、マインスイーパの SX-WINDOW 版。		

No.92	SX-Tatris.x	分類	¥GAME¥TATRIS
作者名	MAPI (小野塚 正則)		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	Ko-Window 上で動く「Tatris」(一世を風靡した「コラムス」と同じルールのゲーム) の SX-WINDOW 版。		

No.93	G_atelier.X	分類	¥GRAPH¥GATELIER
作者名	GOMO		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	JPEG 形式の高速ローダを内蔵したグラフィックエディタ。		
備考	65,536 色モードで IVM.X が常駐していること。		

No.94	GLMpatch.x	分類	¥GRAPH¥GLMPAT
作者名	BLACK.		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	IVM.LB (1994/05/30 12:00 版) に登録されている、GLM リソースで 512×512 ドットを超える画像が扱えない不具合を修正するプログラム。		
備考	実質的な画像サイズの制限は搭載メモリによる。		

No.95	grroot.x	分類	¥GRAPH¥GRROOT
作者名	An		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW のデスクトップの壁紙を 8192 倍 (従来比) 派手にするためのプログラム。		
備考	あらかじめ画面モードを 65,536 色モードに設定し、ビデオマネージャ (IVM.X, IVM.LB) を常駐させ、さらにコントロールパネルの背景設定で「ユーザー」が選択されている必要がある。		

No.96	gr_sel.x	分類	¥GRAPH¥GRSEL
作者名	かごや		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	ビデオマネージャ (IVM.X, IVM.LB) でサポートされている画像ファイルを、grroot.x を使ってランダムに壁紙として設定するプログラム。		
備考	grroot.x ver.1.11 以上 (An 氏作) が必要。		

No.97	壁紙動画.r	分類	¥GRAPH¥HEKIDO
作者名	Sho–Ta		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	SX–WINDOW の壁紙上でスクロール機能のみを使ったアニメーション (書き換えなし) を行うプログラム。		
備考	SX-BASIC 対応。chadance.pic は含まれていません。		

No.98	壁動玉々.r	分類	¥GRAPH¥HEKITM
作者名	Sho–Ta		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	壁紙動画.r (Sho–Ta 氏作) を利用して、SX–WINDOW の壁紙上で多数の青い玉のアニメーションを行うプログラム。		

No.99	vmif-mag.r, vmec-mag.r	分類	¥GRAPH¥IVMAG
作者名	Amalin		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、鮪 CG ファイル形式 (拡張子 .MAG、まきちゃん CG ファイル形式 .MKI の改良版のファイル) を表示するためのインストールスクリプト。		
備考	オリジナルインストーラによりインストール。MAG 形式のセーブは不可。		

No.100	maki_if.r, maki_ec.r	分類	¥GRAPH¥IVMAKI
作者名	Amalin, 西橋		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、まきちゃん CG ファイル形式 (拡張子 .MKI: MAKIchan Graphic の略。鮪 BBS (旧まきちゃんネット) で開発された汎用画像フォーマット) のファイルを読み書きするためのインストールスクリプト。		
備考	オリジナルインストーラによりインストール。		

No.101	PiVM.X	分類	¥GRAPH¥PIVM
作者名	けんと		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	スタートアップメンテ.X に登録することにより、Pi 形式の画像をキャンバス.X や grroot.x (An 氏作) などで扱うことができるようにするプログラム。		
備考	IVM.LB にインストールすることも可能。		

No.102	SxPANIC.r	分類	¥GRAPH¥SXPANIC
作者名	Sho-Ta		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上から、PANIC.x (ぱこたん氏作) によって表示される panic データ (アニメーション) を楽しむためのプログラム。		
備考	画面復帰には SAdjust.r (Sho-Ta 氏作) が必要。		

No.103	SXPIconv.x	分類	¥GRAPH¥SXPICONV
作者名	BLACK.		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	PIX 形式の画像ファイルと圧縮効率のよい Pi 形式 (やなぎさわ氏作) の画像ファイルとの間で相互にデータをコンバートするプログラム。		
備考	ビデオマネージャ (IVM.X, IVM.LB) に追加登録する Pi リソース (けんと氏作) もあり。		

No.104	SXpixpi.x	分類	¥GRAPH¥SXPIXPI
作者名	BLACK.		
対応 OS	SX 3.0/3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	65,536 色モード時に、GRW.X を利用して PIX, Pi 形式の画像を表示するプログラム。		
備考	SX-BASIC 対応。		

No.105	vmcut.INS, vmcut.LB	分類	¥GRAPH¥VMCUT
作者名	BLACK.		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、CUT ファイル (BEEPs 氏作) 形式のファイルを読み書きするためのインストールスクリプト。		
備考	インストーラ.X が必要。		

No.106	VMGIF.INS, VMGIF.LB	分類	¥GRAPH¥VMGIF
作者名	KUM		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、GIF89a に準拠した GIF 形式のファイルを読み書きするためのインストールスクリプト。		
備考	インストーラ.X が必要。		

No.107	vmp2.INS, vmp2.LB	分類	¥GRAPH¥VMP2
作者名	BLACK.		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、PIC2 (やなぎさわ氏作) 形式のファイルを読み書きするためのインストールスクリプト。		
備考	インストーラ.X が必要。		

No.108	VMIF_SC8 VMEC_SC8	分類	¥GRAPH¥VMSC8
作者名	VSYNC		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、SC8 (MSX2 以降の SCREEN8 の VRAM イメージを BSAVE フォーマットで保存したもの) 形式のファイルを読み書きするためのコードリソースバイナリプログラム。		
備考	リソースエディタ、リソースリンカが必要。		

No.109	vmzau.INS, vmzau.LB	分類	¥GRAPH¥VMZAU
作者名	BLACK.		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に追加登録して、シャープ製電子手帳 ZAURUS の手書きメモ (拡張子は .ZAU) を読み書きするためのインストールスクリプト。		
備考	インストーラ.X が必要。		

No.110	Xbmp.x	分類	¥GRAPH¥XBMP
作者名	Ryu!		
対応 OS	SX 3.1 / Human68k	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	MS-Windows および OS/2 で標準フォーマットとなっている「BMP」形式 (無圧縮 .BMP、RLE 圧縮 .RLE、画面比に依存しないもの .DIB) のファイルのローダプログラム。シャーペン.X 上のコンソールからの呼び出しにも対応している。		

No.111	SXlisp.x	分類	¥LANG¥SXLISP
作者名	沖@沖		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で動作する Lisp インタープリタ。		

No.112	iconwdef.r	分類	¥LIB¥ICONWDEF
作者名	HIDORI		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上でのアイコン化をスマートに実現するためのオリジナルウィンドウ定義関数 (WDEF) プログラム。		
備考	サンプルプログラム同梱。		

No.113	moCDEF.r	分類	¥LIB¥MOCDEF
作者名	沖@沖		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	標準ボタンと差し替えることが可能な、立体ボタンのリソースプログラム。		
備考	リソースリンカ (rlk.x, rsc.x, arlk.x など) が必要。		

No.114	picbttn.r	分類	¥LIB¥PICBTTN
作者名	HIDORI		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	沖@沖氏作の moCDEF.r をもとにして picture button に特化したオリジナルコントロール定義関数 “picbttn”。		
備考	サンプルプログラム同梱。		

No.115	SXHEL	分類	¥LIB¥SXHEL
作者名	PRAY		
対応 OS	Human68k	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	Human68k 用のプログラムを SX-WINDOW に移植するためのヘッダ&ライブラリのキット。		
備考	サンプルファイル同梱。		

No.116	TSSLIB	分類	¥LIB¥TSSLIB
作者名	PRAY		
対応 OS	Human68k	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	主にアセンブラで “簡単に” 疑似マルチタスクプログラムを作成するためのライブラリ集。		

No.117	updownlib	分類	¥LIB¥UPDOWN
作者名	けんと		
対応 OS	Human68k	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	数値調整ボタン (アップダウンボタン)、スライドボリューム、テキストエディットの連係処理 (ボタンやスライドボリュームを移動したときに数値を増減させるなど) を楽に行うためのライブラリ。		

No.118	adpplay.r	分類	¥MUSIC¥ADPPLAY
作者名	けんと		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	複数起動可能で、再生周波数の指定が可能な ADPCM ファイルを鳴らすプログラム。		
備考	複数起動には、PCM8.X (江藤氏作) および互換ドライバが必要。		

No.119	SXCDP.X	分類	¥MUSIC¥SXCDP
作者名	結城あすか		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で動作する、CD-ROM ドライブを用いた音楽 CD 再生プログラム。一般の音楽 CD 再生機と同等の機能を備えている。		
備考	CDDEV.SYS (計測技研製 CD-ROM デバイスドライバ) が必要。		

No.120	SXWAVPLAY.X	分類	¥MUSIC¥SXWAV
作者名	Yokko		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で .WAV ファイル、または .AVI ファイル中の音声データを再生するプログラム。		
備考	PCM8.X (江藤氏作) および互換ドライバが必要。		

No.121	SXZC.r	分類	¥MUSIC¥SXZC
作者名	けんと		
対応 OS	SX 3.0 以上 / Z-MUSIC ver.2.00 以上	収録内容	第 3 版 83 刷のソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ZMUSIC.X, rcd.x 等の曲データ (拡張子 .opm, .zmd, .zms, .mdx, .mdn, .mdf, .rcp, .r36, .mcp, .mdz, .mdi, .zdf, .mid, .mff, .std) を再生するプログラム。		
備考	第 3 版 85 刷の差分ファイル同梱。一部データ再生には別途 mzp.x が必要。SX-BASIC 対応。		

No.122	印刷中は演奏停止.r /mzj.x/opm に.x /cm64reveb.sxb	分類	¥MUSIC¥SXZCSUP
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SXZC.r (けんと氏作) のサポートプログラム。		

No.123	WordMaker.X	分類	¥MUSIC¥WORDMK
作者名	結城あすか		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SXCDP.X (結城あすか氏作) で利用できる歌詞表示用のデータ作成支援プログラム。		
備考	cddev.sys (計測技研製 CD-ROM デバイスドライバ) が必要。		

No.124	SXP1.X	分類	¥TERM¥BPLUS
作者名	けんと		
対応 OS	SX 3.1	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	Communication SX-68K と QuTERM で使用可能な、B-Plus プロトコル転送プログラム。		

No.125	コンピュータ画面.r	分類	¥TERM¥CZCOMP
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	QuTERM.X でリダイアル中にテレビを見ることができるプログラム。		
備考	専用ディスプレイ TV が必要 (CZ-6XX シリーズ)。		

No.126	gnus.el/ほか	分類	¥TERM¥GNUS
作者名	BAA		
対応 OS	SX 3.1 / Human68k 3.02	収録内容	lisp マクロ
サポートネット	Network-SX NG		
紹介	Mule や Nemacs 上でインターネット経由のニュースを読むためのニュースリーダ。		

No.127	SXP2.X	分類	¥TERM ¥MLINK
作者名	PRAY		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	Communication SX-68K と QuTERM で使用可能な MLINK プロトコル転送プログラム。		

No.128	nr.x	分類	¥TERM¥NR
作者名	Mew		
対応 OS	SX 3.1	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	ログファイルを見ながら、アーティクルのポスト、フォロー、メールの送受信を一括で行えるようにするための、オフライン・ニューズリーダープログラム。		

No.129	QINT.x	分類	¥TERM¥QINT
作者名	PRAY		
対応 OS	Human68k	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 利用時にマウス関係の割り込み負荷を軽くすることで、X680x0 の処理性能をほんのわずかだけ向上させ、X68000 XVI 以下 (MPU68000 の機種) での高速通信 (QuTERM 支援) に対応 (DTE38400) させるプログラム。		
備考	X68030 以上 (MPU68030, 68040) にも対応。		

No.130	QuTERM.X	分類	¥TERM¥QUTERM
作者名	George		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX-WINDOW 上で動作する、LISP インタープリタを内蔵した究極のターミナルソフトのフルセットバージョン。NIFTY-Serve や Network-SX NG のオートログインが可能。		
備考	RSDRV ver.2.00 以上もしくは TMSIO.X の常駐が必要。		

No.131	QuTERM_S.X	分類	¥TERM¥QUTERMS
作者名	George		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	QuTERM.X のスモールセットバージョン。メインメモリ 2M バイトでも起動が可能。		
備考	RSDRV ver.2.00 以上もしくは TMSIO.X の常駐が必要。		

No.132	sxqv.x	分類	¥TERM¥SXQV
作者名	BLACK.		
対応 OS	SX 3.0/3.1	収録内容	別途ソースファイル同梱
サポートネット	Network–SX NG		
紹介	PC–VAN 等でサポートされている Quick-VAN のプロトコル転送を行うプログラム。Communication SX-68K、QuTERM 等で利用可能。		
備考	RSDRV.SYS ver.2.0 以上で動作 (TMSIO では不可)。ターミナルソフトから呼ばれることが前提条件。		

No.133	autocase.x	分類	¥TOOL¥AUTOCASE
作者名	GRANADA		
対応 OS	SX 1.1 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	フロッピーディスク内のすべてのファイル名、ディレクトリ名を自動的に大文字にリネームするプログラム。		

No.134	電卓管理.r	分類	¥TOOL¥CALCMAN
作者名	けんと		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	SX–WINDOW 上で [OPT.1]+[OPT.2] キーを押すことにより電卓を起動するためのプログラム。実行オプションの指定によって種々の電卓 (数値演算.x：HI–Br. 氏作等) も起動可能。		

No.135	calcSx.x	分類	¥TOOL¥CALCSX
作者名	仁泉大輔		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	三角関数や変数が使える汎用の計算機プログラム。		

No.136	canvas.r	分類	¥TOOL¥CANVAS
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	ビデオマネージャ (IVM.X, IVM.LB) でサポートされる画像データを 16 色モード、65,536 色モードごとに自動判別して起動するプログラム。		

No.137	CHIDIR!.X	分類	¥TOOL¥CHIDIR
作者名	美姫		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	SX–WINDOW から DOS のカレントディレクトリを変更することができるプログラム。SX–WINDOW 起動時のカレントディレクトリに、シャーペン.X がファイルを作成するのを強制的に変更することができる。		

No.138	クリックノート.x	分類	¥TOOL¥CLNOTE
作者名	高橋博樹 (HI–Br.)		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY–Serve		
紹介	クリックノート.x 上のボタンを押すことによってさまざまなアプリケーション (SX–WINDOW 内部コマンドを含む) を割り振ることが可能なランチャ。1 ページごとに 8 アプリケーションが登録可能で、全部で 8 ページが使用可能。		
備考	SX-BASIC 対応。		

No.139	CLOSEWITHOUT.X	分類	¥TOOL¥CLOSEWITH
作者名	An		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	Network–SX NG		
紹介	SX シェルのシステムアイコンメニューにある「全クローズ」のかわりに、指定したタスク以外を終了するプログラム。		

No.140	COPYBACK.X	分類	¥TOOL¥COPYBACK
作者名	VSYNC		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	SX–WINDOW 上で更新されたファイルを自動的にチェックし、SX–WINDOW 終了時に指定ドライブの指定パスへまとめて転送するプログラム。		
備考	環境ファイル (COPYBACK.ENV) を使って、バックアップをとりたくないファイルやパスをワイルドカードで指定することが可能。		

No.141	CRTCTRL.r	分類	¥TOOL¥CRTCTRL
作者名	Sho–Ta		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	SX–WINDOW の実画面サイズを拡大するための、フルマウスオペレーションの CRTC セットアッププログラム。		
備考	SAdjust.r ver.1.20 (Sho–Ta 氏作) 以降の常駐が必要。SX-BASIC 対応。		

No.142	カップめん.X	分類	¥TOOL¥CUPMEN
作者名	カップめん		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	カップめんを作るときのタイマープログラム。30 秒 ～ 10 分まで、30 秒単位で設定することができる。		

No.143	doc.x	分類	¥TOOL¥DOC
作者名	RICE		
対応 OS	SX 3.1 / Human68k	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	指定のビューア(less.x、シャーペン.X 等複数指定可能)で、拡張子.DOC, .MAN 等のマニュアルを読み込むためのプログラム。		
備考	シャーペン.X 上のコンソールからの呼び出しにも対応。		

No.144	DirViewer.x	分類	¥TOOL¥DV
作者名	Guges		
対応 OS	SX 3.01 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で特定のディレクトリを短時間かつ簡単に表示・オープンすることができるディレクトリビューア。		

No.145	EMAGENCY.r	分類	¥TOOL¥EMAGEN
作者名	Yoz.		
対応 OS	SX 1.1 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	システムイベント ENDTSK を疑似的に作成して、SX シェルを含むすべてのタスクを終了させるプログラム。		

No.146	数値演算.X	分類	¥TOOL¥ENZAN
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	関数グラフ機能付き関数電卓プログラム。		
備考	SX-BASIC 対応。		

No.147	F2SC.x	分類	¥TOOL¥F2SC
作者名	カップめん		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ファイルの内容をデスクトップスクラップに直接コピーすることができるプログラム。		

No.148	GARBAGE.X	分類	¥TOOL¥GARBAGE
作者名	KUM		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	終了したタスクがメモリブロックを未開放のまま残していないかどうかを調べて報告し、必要に応じてそれらを開放するためのプログラム。		

No.149	画面コピー.X	分類	¥TOOL¥GCOPY
作者名	Yokko		
対応OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で画面イメージをクリップボードに転送するプログラム。テキスト画面、グラフィック画面双方に対応。		

No.150	汎用トレイ.x	分類	¥TOOL¥HANTRAY
作者名	PRAY		
対応OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上から実行できる多機能なファイルトレイプログラム。		
備考	いろいろな用途の環境ファイル付属 (*.cnf)。		

No.151	HisClip.X	分類	¥TOOL¥HISCLIP
作者名	高橋博樹 (HI-Br.)		
対応OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW のクリップボードに送られてくるデータのうち、最新の 16 件を保存しておくプログラム。		

No.152	越後屋残量.x	分類	¥TOOL¥HZAN
作者名	越後屋		
対応OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で動作し、メモリやハードディスクなどの補助記憶装置の空き容量をリアルタイムで表示するプログラム。		

No.153	IVMInfo.x	分類	¥TOOL¥IVMINFO
作者名	BLACK.		
対応OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に登録されている画像形式のリソース情報を表示するプログラム。		

No.154	こんなものポイだ.r	分類	¥TOOL¥KONPOI
作者名	けんと		
対応OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	アイコンリスト、アイコンメンテなどで設定することにより、ファイル (複数も可) をダブルクリックするだけでクリーナに捨てることができるプログラム。		

No.155	MFOCK.X	分類	¥TOOL¥MFOCK
作者名	VSYNC		
対応OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で、一度に複数のファイルを実行するためのプログラム。		

No.156	Mini.x	分類	¥TOOL¥MINI
作者名	Guges		
対応 OS	SX 2.0 以上	収録内容	バイナリのみ
サポートネット	NIFTY-Serve		
紹介	デスクトップ上のウィンドウの配置を縮小して表示し、それぞれにクローズ、アクティベート等のコントロールを行うことができるプログラム。		

No.157	文字列検索.x	分類	¥TOOL¥MOJI
作者名	ウィル		
対応 OS	SX 3.0 以上	収録内容	別途ソース同梱
サポートネット	NIFTY-Serve		
紹介	正規表現検索等をサポートした高機能な文字列検索プログラム。		

No.158	NDS.X	分類	¥TOOL¥NDS
作者名	PRAY		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	シャーペン.X の文書保存のデフォルトのファイル名 (TEXT?.PEN) を、任意のファイル名に変更するプログラム。		

No.159	新パス名.X	分類	¥TOOL¥NEWPATH
作者名	特α級電脳師		
対応 OS	SX 3.0	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	ファイル名、パス名、フルパス名を表示したり、表示したファイル名等をクリップボードに転送したりするプログラム。		

No.160	ぬるぬる.X	分類	¥TOOL¥NURU
作者名	カップめん		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ファイル中のヌル記号 (NULL) を取り除き、CR/LF を CRLF に変換し、タブコードをスペースコードに変換するプログラム。		

No.161	OpenDir!.x	分類	¥TOOL¥OPENDIR
作者名	かごや		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	深く階層化したディレクトリを、SX-WINDOW 上で短時間かつ簡単に表示・オープンするプログラム。		

No.162	path.x	分類	¥TOOL¥PATH
作者名	Eban.W		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上でドラッグされたファイルアイコンのパス名を表示し、クリップボードに転送するプログラム。		

No.163	PICICON.X	分類	¥TOOL¥PICCON
作者名	VSYNC		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で IVM.X が扱える画像データを縮小・減色処理することによって PAT4 形式の画像データに変換し、同時に ICON.LB へアイコンを自動登録 (もしくは削除) するためのツール。		
備考	IVM.X が常駐していること。		

No.164	ps.x, kill.x	分類	¥TOOL¥PSKILL
作者名	かごや		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	シャーペン.X のコンソール上で動くプログラム。ps.x はプロセスを表示し、kill.x はタスクの終了を行うもの。		

No.165	pt4get.x	分類	¥TOOL¥PT4GET
作者名	BAA		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	シャーペン形式のファイル (アイコンが埋め込まれたドキュメント等) から PAT4 形式のデータをドラッグしたり、メニューを使って PAT4 形式のデータのオープン、スクラップへのコピー、ICON.LB への登録が可能なプログラム。		

No.166	PICTtoDRAW.x	分類	¥TOOL¥PICTDRAW
作者名	高橋博樹 (HI-Br.)		
対応 OS	SX 2.0 以上	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	クリップボードにある PICT データを Easydraw.X で利用可能な DRAW データに変換するプログラム。		
備考	シャーペン.X に張り付けることはできない。		

No.167	regsea.ex	分類	¥TOOL¥REGSEA
作者名	けんと		
対応 OS	SX-3.1	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	正規表現 [regexp] にマッチする部分を検索する、シャーペン.X の外部コマンド。		
備考	SX-WINDOW ver.3.1 付属のシャーペン.X が必要。		

No.168	名前変更.x	分類	¥TOOL¥REN
作者名	カップめん		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ファイル名やボリューム名の変更・複製、新規ディレクトリの作成や大文字小文字化、ノード・拡張子別のキャピタライズが変更可能なプログラム。		

No.169	IVM モジュール削除.x	分類	¥TOOL¥RMIVM
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	ビデオマネージャ (IVM.X, IVM.LB) に登録されている画像モジュールを登録解除するプログラム。		

No.170	roottoscrap.x	分類	¥TOOL¥ROOTCOPY
作者名	An		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	grroot.x (An 氏作) 等で表示されたグラフィック画面をスクラップにコピーするためのプログラム。		
備考	SxPANIC.r (Sho-Ta 氏作)、SAdjust.r (Sho-Ta 氏作) が必要。		

No.171	SCopy.x	分類	¥TOOL¥SCOPY
作者名	カップめん		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	実画面全体をデスクトップスクラップに PAT4 形式でコピーするプログラム。		

No.172	SendMes.x	分類	¥TOOL¥SEND
作者名	カップめん		
対応 OS	SX 3.0 以上	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	SX-BASIC 形式で、いろいろなタスクにメッセージを送るプログラム。		

No.173	SMAutoexec.x	分類	¥TOOL¥SMAUTO
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ストリームマネージャ (streamMan.x：ともみ氏作) を使用してタスクの自動起動とポート間の自動接続を行うプログラム。		
備考	streamMan.x ver.0.3 以上 (ともみ氏作) が必要。		

No.174	sxBack.x	分類	¥TOOL¥SMBACK
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	スクラップとストリームマネージャ (streamMan.x：ともみ氏作) によって提供される、ストリームによって壁紙設定を行うプログラム。		
備考	streamMan.x ver.0.4 以上 (ともみ氏作) が必要。		

No.175	sxCut.x	分類	¥TOOL¥SMCUT
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	スクラップとストリームマネージャ (streamMan.x：ともみ氏作) によって提供される、ストリームからの画像を読み込んで、その一部を切り出すプログラム。		
備考	streamMan.x ver.0.4 以上 (ともみ氏作) が必要。		

No.176	sxGallery.x	分類	¥TOOL¥SMGAL
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	ストリームマネージャ (streamMan.x： ともみ氏作) 対応のビデオマネージャ (IVM.X, IVM.LB) 対応画像データを表示するプログラム。		
備考	streamMan.x ver.0.4 以上 (ともみ氏作) が必要。また、IVM.X が常駐していること。		

No.177	sxGFrame.x	分類	¥TOOL¥SMGF
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	65,536 色モードでのスクラップとストリームマネージャ (streamMan.x：ともみ氏作) によって提供される、ストリームからの画像ビューア。		
備考	streamMan.x ver.0.3 以上 (ともみ氏作) が必要。		

No.178	scst.x	分類	¥TOOL¥SMSCST
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	スクラップとストリームマネージャ (streamMan.x：ともみ氏作) によって提供される、ストリーム間のデータ交換を行うプログラム。		
備考	streamMan.x ver.0.4 以上 (ともみ氏作) が必要。		

No.179	streamMan.x	分類	¥TOOL¥SMSTREAM
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	統一的なタスク間通信とタスク間接続確立のためのユーザインターフェースを提供するプログラム。		
備考	ストリームマネージャ用のライブラリ付属。		

No.180	sxg2t.x	分類	¥TOOL¥SMSXG2T
作者名	ともみ		
対応 OS	SX 3.0 以上	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	ストリームマネージャ (streamMan.x：ともみ氏作) を利用して、入力された 65,536 色/16 色ビットマップデータをプロットイメージに変換するフィルタプログラム。		
備考	streamMan.x ver.0.4 以上、sxGallery.x, scst.x (いずれも ともみ氏作) などのストリーム入出力プログラムと 4M バイト以上のメインメモリが必要。		

No.181	SXAutoDir.x	分類	¥TOOL¥SXAD
作者名	PRAY		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	ディスク挿入時に、ディスク内のディレクトリをオープンさせるプログラム。		

No.182	SXBdif.x	分類	¥TOOL¥SXBDIF
作者名	PRAY		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	バイナリ差分ファイル作成プログラム BDIF.X (ひがしで氏作) の SX–WINDOW 版プログラム (拡張子は.BFD)。		
備考	タスクごとのメモリ消費量が大きいラージモデル差分同梱。FSX.X ver.3.00 以降用。		

No.183	SXBJ.X	分類	¥TOOL¥SXBJ
作者名	カップめん		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network–SX NG/NIFTY–Serve		
紹介	SX–WINDOW 上で、ESC/P24 対応プリンタや CZ 系列のプリンタを使ってマルチタスクでリストアウト (文字のみ) するプログラム。		

No.184	SXBup.x	分類	¥TOOL¥SXBUP
作者名	PRAY		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	バイナリ差分ファイルアップデートプログラム BUP.X (ひがしで氏作) の SX–WINDOW 版プログラム (拡張子は .BFD)。		
備考	タスクごとのメモリ消費量が大きいラージモデル差分同梱。		

No.185	SX_CRLF.X	分類	¥TOOL¥SXCRLF
作者名	PRAY		
対応 OS	SX 3.1	収録内容	別途ソースファイル同梱
サポートネット	Network–SX NG		
紹介	クリップボード中のテキストの改行を CRLF (MS–DOS 形式) に変更するフィルタプログラム。		

No.186	sx_grep.x	分類	¥TOOL¥SXGREP
作者名	伊能富士夫		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上でファイル中の指定文字列を検査するプログラム。awk 互換の正規表現が利用可能。		

No.187	SXjis.x	分類	¥TOOL¥SXJIS
作者名	仁泉大輔		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	EUC, SJIS, JIS ファイルの漢字コードを変換するプログラム。		

No.188	SXman.x	分類	¥TOOL¥SXMAN
作者名	An		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW 上で、拡張子.PEN，.DOC，.MAN 等のマニュアルおよびドキュメントの検索や表示・管理などを行うプログラム。		
備考	マニュアルを表示するためのプログラム(シャーペン.X 等)が必要。		

No.189	SXmic.x	分類	¥TOOL¥SXMIC
作者名	PRAY		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network-SX NG		
紹介	バイナリテキストコンバータ mic (milk. 氏作成の ish コンバータ) の SX-WINDOW 版。クリップボードを入出力先に指定することが可能。		

No.190	sxmode.x	分類	¥TOOL¥SXMODE
作者名	BLACK.		
対応 OS	SX 3.0/3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG		
紹介	SX-WINDOW ver.3.0 以前の旧 16 色モード用に作られたグラフィック関係のソフトが 65,536 色モードで暴走するのを未然に防ぐためのプログラム。現在の表示色数を表示する。		

No.191	momocopy.x	分類	¥TOOL¥SXMOMO
作者名	亜山 雪		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	ハードディスクをテンポラリ領域として、1 ドライブで MO ディスクのコピーが可能なプログラム。		

No.192	`SXMP.x`	分類	¥TOOL¥SXMP
作者名	カップめん		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	マウスポインタのグローバル座標、任意のウィンドウを基準にしたローカル座標を表示するプログラム。		

No.193	`SXPerform.X`	分類	¥TOOL¥SXPFM
作者名	Yokko		
対応 OS	SX 2.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	SX-WINDOW 上で一定時間 (約 3 秒) ごとにパフォーマンス値を測定して表示したり、X68030 以上 (MPU68030, 68040) の機種のキャッシュの設定と動作ウェイトを設定するプログラム。		

No.194	`SxSED.X`	分類	¥TOOL¥SXSED
作者名	仁泉大輔		
対応 OS	SX 3.0 以上	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	漢字対応 `sed (Stream EDitor)` ver.1.5 Rel.8 の SX-WINDOW 版。		

No.195	`sxsh.x`	分類	¥TOOL¥SXSH
作者名	ドロネ		
対応 OS	SX 3.1	収録内容	別途ソースファイル同梱
サポートネット	NIFTY-Serve		
紹介	アイコンメンテやメニューメンテなど、実行ファイルの設定で複数のコマンドを起動したり、コンソール上で実行するコマンドを直接指定することができる簡易シェルプログラム。		

No.196	窓を後へ`.r`/窓を前へ`.r`/窓を操る`.r`	分類	¥TOOL¥WINMAN
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	クリックノート`.x` (HI-Br. 氏作) からキーボードを使ってアクティブウィンドウを移動するプログラム。		

No.197	窓を動かす`2.x`	分類	¥TOOL¥WINMOV
作者名	けんと		
対応 OS	SX 3.1	収録内容	ソースファイル同梱
サポートネット	Network-SX NG/NIFTY-Serve		
紹介	クリックノート`.x` (HI-Br. 氏作) からキーボードを使ってアクティブウィンドウを移動するプログラムその 2。ウィンドウ枠の色が変更可能。		

No.198	WinThief.x	分類	¥TOOL¥WINTHIEF
作者名	T·A·N		
対応 OS	SX 3.1	収録内容	バイナリのみ
サポートネット	Network–SX NG		
紹介	アクティブウィンドウのイメージを取り込み、クリップボードに PAT4 形式で転送するプログラム。		

注: Network–SX NG Tel. 03-5603-7197 (事情によって電話番号が変わる可能性があります。)

第2部

SXコール
リファレンス

SXコールリファレンスの利用法

本リファレンスでは、SXコールはマネージャ単位で、原則としてSXコール番号順に並んでいます。また、アセンブラレベルでの各SXコールに渡す引数、返り値、機能などのほかに、C言語で記述する場合の関数についても解説しています。C言語の関数については、関数名、引数と型、返り値の型とその意味について解説を行っています。

SXコール以外では、C言語用のライブラリとしてのみ用意されている関数やマクロについてもふれています。

凡例

●$A450
SXコール番号。
SXコール番号のかわりに[ライブラリ]と表示されているものは、ライブラリとしてのみ用意されている関数を意味する。[マクロ]とあるものは、マクロとしてのみ用意されていることを意味する。

●SXPack
コール名。

●3.1
2.0 と表示されているSXコールはSX-WINDOW ver.2.0から追加されたことを意味している。同様に 3.0 、 3.1 と表示されているSXコールはそれぞれSX-WINDOW ver.3.0、ver.3.1から追加されたことを意味している。

●引数
コールする際にスタックに積むべき引数。そのサイズ、引数名、引数の持つ意味が書かれている。

●返り値
呼び出し後の返り値。レジスタとその意味が書かれている。

●機能
コール機能解説。そのコールが持つ働きと使用上の注意が書かれている。

$A450　　SXPack　　3.1

引　数► word　pID　；データ圧縮定義関数のID
long　srcLen　；圧縮前のデータのバイト数
long　destPtr　；圧縮後のデータを格納するバッファへのポインタ
long　srcPtr　；圧縮前のデータへのポインタ

返り値► D0.L　圧縮後のデータのサイズ/リザルトコード
A0.L　destPtr として 0 を指定した場合、データを格納したハンドル

機　能► srcPtr で指定した srcLen バイトのデータを、pID で指定したデータ圧縮定義関数によって圧縮する。
結果は destPtr で指定したバッファに格納される。destPtr として 0 を指定した場合、データ圧縮定義関数が再配置可能ブロックを確保し、そのなかで圧縮後のデータを格納する。このハンドルは A0.L に返される。
再配置が発生する。

Cの関数► int *SXPack*(short pID, long srcLen, void *destPtr, void *srcPtr, Handle *destHdl);
destPtr がNULLの場合、圧縮データのハンドルはハンドル destHdl に格納される。
返り値は圧縮後のデータのサイズまたはリザルトコード。

●Cの関数
Cのプログラムから SX コールを呼び出す場合の関数定義、および返り値についての説明。関数定義では、それぞれの引数の意味がわかるように、[引数]の欄の名前と同一の名前を使用している。
返り値としては、原則として[返り値]で示したD0.Lの内容と同じものが返るが、異なる場合もある。また、D0とA0の両方に値が返るコールの場合、引数として渡した変数へのポインタに返り値が返る場合がある。
原則として Workroom ＋ SX31KIT 環境に準じている。

メモリマン #include <MEMORY.H>

$A000 MMInitHeap

引　数► long IZRecord ；レコードのアドレス
返り値► D0.L リザルトコード
機　能► **使用禁止コール。**
$A00E MMHeapInit と同様。
パラメータはレコード型。

IZRecord:
```
        long    startPtr      ;ヒープゾーン先頭アドレス
        long    limitPtr      ;ヒープゾーン終端アドレス
        word    cMoreMasters  ;マスタポインタ数
        long    pGrowZone     ;グローヒープゾーン関数のアドレス
```

このコールによって初期化されたヒープゾーンは、メモリ初期化省略フラグがTRUEになる。

$A001 MMGetCurrentHeap

引　数► なし
返り値► D0.L リザルトコード
A0.L カレントヒープゾーンの先頭アドレス
機　能► **使用禁止コール。**
$A01C MMChGet と同様。

$A002 MMSetCurrentHeap

引　数► long hz ；ゾーンヘッダの先頭アドレス
返り値► D0.L リザルトコード
機　能► **使用禁止コール。**
$A01D MMChSet と同様。

$A003　MMNewHandle

引　数► long　logicalSize　；作成するブロックの論理サイズ
返り値► D0.L　リザルトコード
　　　　A0.L　作成されたブロックへのハンドル
機　能► **使用禁止コール。**
　　　　$A021 MMChHdlNew と同様。

$A004　MMSetHandleSize

引　数► long　h　；サイズを変更するブロックへのハンドル
　　　　long　newSize　；新しい論理サイズ
返り値► D0.L　リザルトコード
　　　　A0.L　ハンドルがそのまま返る
機　能► **使用禁止コール。**
　　　　$A03A MMHdlSizeSet と同様。

$A005　MMDisposeHandle

引　数► long　h　；廃棄するブロックへのハンドル
返り値► D0.L　リザルトコード
　　　　A0.L　ハンドルがそのまま返る
機　能► **使用禁止コール。**
　　　　$A038 MMHdlDispose と同様。

$A006　MMGetHandleSize

引　数► long　h　；サイズを得るブロックへのハンドル
返り値► D0.L　論理サイズ/リザルトコード
　　　　A0.L　ハンドルがそのまま返る
機　能► **使用禁止コール。**
　　　　$A039 MMHdlSizeGet と同様。

$A007　MMHLock

引　数► long　h　；ロックするブロックへのハンドル
返り値► D0.L　リザルトコード
　　　　A0.L　ハンドルがそのまま返る

機　能► 使用禁止コール。
$A040 MMHdlLock と同様。

$A008　　MMHUnlock

引　数► long　h　;ロックを解除するブロックへのハンドル
返り値► D0.L　リザルトコード
A0.L　ハンドルがそのまま返る
機　能► 使用禁止コール。
$A041 MMHdlUnlock と同様。

$A009　　MMNewPtr

引　数► long　logicalSize　;作成するブロックの論理サイズ
返り値► D0.L　リザルトコード
A0.L　作成されたブロックへのポインタ
機　能► 使用禁止コール。
$A01E MMChPtrNew と同様。

$A00A　　MMDisposePtr

引　数► long　p　;廃棄するブロックへのポインタ
返り値► D0.L　リザルトコード
A0.L　ポインタがそのまま返る
機　能► 使用禁止コール。
$A02F MMPtrDispose と同様。

$A00B　　MMGetPtrSize

引　数► long　p　;サイズを得るブロックへのポインタ
返り値► D0.L　論理サイズ/リザルトコード
A0.L　ポインタがそのまま返る
機　能► 使用禁止コール。
$A030 MMPtrSizeGet と同様。

$A00C　　MMSetPtrSize

引　数► long　p　;サイズを変更するブロックへのポインタ

long　newSize　；新しい論理サイズ
返り値► D0.L　リザルトコード
A0.L　ポインタがそのまま返る
機　能► 使用禁止コール。
$A031 MMPtrSizeSet と同様。

$A00D　MMCompactMem

引　数► long　cbNeeded　；必要な空きスペース
返り値► D0.L　フリーブロックのアドレス
A0.L　フリーブロックのアドレス
機　能► 使用禁止コール。
$A010 MMMemCompact と同様。

$A00E　MMHeapInit

引　数► long　startPtr　；ヒープゾーン先頭アドレス
long　limitPtr　；ヒープゾーン終端アドレス
long　cMoreMasters　；マスタポインタの数
long　pGrowZone　；グローヒープゾーン関数のアドレス
long　vis　；メモリ初期化省略フラグ
返り値► D0.L　= 0　エラー
≠ 0　ゾーンヘッダのアドレス
機　能► startPtr から limitPtr-1 までのヒープゾーンを生成する。
cMoreMasters は 0 ～ $7FFF。これ以外の場合、$40 が指定される。これは、1つのマスタポインタブロックのなかにマスタポインタをいくつ収めるかを意味している。
pGrowZone で指定したグローヒープゾーン関数は、このヒープゾーンを拡大する際に呼ばれるルーチンのアドレス。pGrowZone に 0 を指定すると、デフォルトのルーチンが使用される。
vis はメモリ初期化省略フラグで、これに FALSE (= 0) を指定した場合、このヒープゾーンからメモリブロックを確保/開放する際に初期化を行う。
Cの関数► Heap **MMHeapInit*(void *startPtr, void *limitPtr, int cMoreMasters, int (*pGrowZone)(Heap *hz, long cbNeeded), BOOLEAN vis);
返り値はゾーンヘッダのアドレス。

$A00F　MMBlockMstGet

引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　p　；マスタポインタを求めるブロックヘッダのアドレス

返り値► D0.L　マスタポインタ/リザルトコード

機　能► p で指定した再配置可能ブロックのマスタポインタを返す。p はブロックヘッダのアドレスであることに注意。

Cの関数► Master *MMBlockMstGet(Heap *hz, Block *p);
返り値はマスタポインタ/リザルトコード。

$A010　MMMemCompact

引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　cbNeeded　；必要な空きスペース

返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　フリーブロックへのポインタ

機　能► hz で指定したヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、ブロックの再配置を繰り返す。

Cの関数► Block *MMMemCompact(Heap *hz, long cbNeeded);
返り値はフリーブロックへのポインタ。

$A011　MMMemPurge

引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　cbNeeded　；必要な空きスペース

返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　フリーブロックへのポインタ

機　能► hz で指定したヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、パージ可能ブロックのパージを繰り返す。

Cの関数► Block *MMMemPurge(Heap *hz, long cbNeeded);
返り値はフリーブロックへのポインタ。

$A012　MMMemMelt

引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　cbNeeded　；必要な空きスペース

返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　フリーブロックへのポインタ

機　能► hz で指定したヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、再配置可能ブロックの再配置とパージ可能ブロックのパージを繰り返す。

Cの関数► Block **MMMemMelt*(Heap *hz, long cbNeeded);
返り値はフリーブロックへのポインタ。

$A013　MMMemReserve

引　数► long　hz　　　　　; ヒープゾーンのアドレス
　　　　long　cbNeeded　　; 必要な空きスペース

返り値► D0.L　= 0　エラー
　　　　　　　$\neq$ 0　フリーブロックへのポインタ

機　能► hz で指定したヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、再配置可能ブロックの再配置とパージ可能ブロックのパージを繰り返す。
このコールでは、できるだけ下位にフリーブロックを作成する。そのため、ロックして使用するブロックを確保する前にこのコールを呼び出しておくと、再配置が効率よく行われる。

Cの関数► Block **MMMemReserve*(Heap *hz, long cbNeeded);
返り値はフリーブロックへのポインタ。

$A014　MMMemSizeFree

引　数► long　hz　　　　　; ヒープゾーンのアドレス

返り値► D0.L　最大のフリーブロックのサイズ

機　能► hz で指定したヒープゾーンのなかの最大のフリーブロックのサイズを返す。

Cの関数► long *MMMemSizeFree*(Heap *hz);
返り値はフリーブロックのサイズ。

$A015　MMMemSizeComp

引　数► long　hz　　　　　; ヒープゾーンのアドレス

返り値► D0.L　ブロックの物理サイズ

機　能► hz で指定したヒープゾーンで再配置を行った場合に得られるであろう最大のフリーブロックの物理サイズを返す。実際に再配置が行われるわけではない。

Cの関数► long *MMMemSizeComp*(Heap *hz);
返り値はブロックの物理サイズ。

$A016　MMMemSizePurg

引　数► long　hz　；ヒープゾーンのアドレス
返り値► D0.L　ブロックの物理サイズ
機　能► hzで指定したヒープゾーンでパージを行った場合に得られるであろう最大のフリーブロックの物理サイズを返す。実際にパージが行われるわけではない。
Cの関数► long *MMMemSizePurg*(Heap *hz);
返り値はブロックの物理サイズ。

$A017　MMMemSizeMelt

引　数► long　hz　；ヒープゾーンのアドレス
返り値► D0.L　ブロックの物理サイズ
機　能► hzで指定したヒープゾーンで再配置とパージを行った場合に得られるであろう最大のフリーブロックの物理サイズを返す。実際に再配置やパージが行われるわけではない。
Cの関数► long *MMMemSizeMelt*(Heap *hz);
返り値はブロックの物理サイズ。

$A018　MMMemErrorGet

引　数► なし
返り値► D0.L　リザルトコード
機　能► メモリマンで最後に発生したリザルトコードを得る。
Cの関数► int *MMMemErrorGet*(void);
返り値はリザルトコード。

$A019　MMMemErrorSet

引　数► long　resultCode　；リザルトコード
返り値► D0.L　前のリザルトコード
機　能► リザルトコードをセットする。
Cの関数► int *MMMemErrorSet*(int resultCode);
返り値は前のリザルトコード。

$A01A　MMMemStrictGet

引　数► なし
返り値► D0.L　厳密チェックフラグの内容

機　能► 厳密チェックフラグを得る。

$A01B　　MMMemStrictSet

引　数► long　strictFlag　　;厳密チェックフラグ
返り値► D0.L　前の厳密チェックフラグ
機　能► 厳密チェックフラグを設定する。
厳密チェックフラグを TRUE にすると、メモリマンに渡すハンドルやポインタは通常よりも厳しくチェックされ、不当な値の場合はエラーを返すようになる。

$A01C　　MMChGet

引　数► なし
返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　カレントヒープゾーンの先頭アドレス
機　能► カレントヒープゾーンの先頭アドレス（ゾーンヘッダのアドレス）を返す。
Cの関数► Heap **MMChGet*(void);
返り値はヒープゾーンの先頭アドレス。

$A01D　　MMChSet

引　数► long　hz　　;ヒープゾーンのアドレス
返り値► D0.L　= 0　正常終了
　　　　　　　≠ 0　エラー
機　能► カレントヒープゾーンを hz にする。
Cの関数► int *MMChSet*(Heap *hz);
返り値はエラーの有無。

$A01E　　MMChPtrNew

引　数► long　logicalSize　　;作成するブロックの論理サイズ
返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　作成されたブロックへのポインタ
機　能► logicalSize の論理サイズを持つ再配置不能ブロックをカレントヒープゾーン中に作成する。
これによって作成されるブロックの属性は、

- 再配置不能
- リソースビット OFF

という状態になる。
再配置が発生する。

Cの関数► Pointer *MMChPtrNew*(long logicalSize);
返り値はブロックへのポインタ。

$A01F　MMChMstMore

引　数► なし

返り値► D0.L　= 0　正常終了
　　　　　　≠ 0　エラー

機　能► マスタポインタブロックを追加作成する。
再配置が発生する。

Cの関数► int *MMChMstMore*(void);
返り値はエラーの有無。

$A020　MMChMstNew

引　数► なし

返り値► D0.L　= 0　エラー
　　　　　　≠ 0　新しいハンドル

機　能► 新しい空のハンドルを返す。メモリの確保などは行わない。
再配置が発生する。

Cの関数► Master **MMChMstNew*(void);
返り値は新しいハンドル。

$A021　MMChHdlNew

引　数► long　logicalSize　; 作成するブロックの論理サイズ

返り値► D0.L　= 0　エラー
　　　　　　≠ 0　作成されたブロックへのハンドル

機　能► logicalSize の論理サイズを持つ再配置可能ブロックをカレントヒープ中に作成する。
これによって作成されるブロックの属性は、

- 再配置可能
- ロックされていない
- パージ不可
- リソースビット OFF

という状態になる。
再配置が発生する。

Cの関数► _Handle *MMChHdlNew*(long logicalSize);
返り値はブロックへのハンドル。

$A022　MMChCompact

引　数► long　cbNeeded　;必要な空きスペース

返り値► D0.L　= 0　エラー
≠ 0　フリーブロックへのポインタ

機　能► カレントヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、ブロックの再配置を繰り返す。

Cの関数► Block **MMChCompact*(long cbNeeed);
返り値はフリーブロックへのポインタ。

$A023　MMChPurge

引　数► long　cbNeeded　;必要な空きスペース

返り値► D0.L　= 0　エラー
≠ 0　フリーブロックへのポインタ

機　能► カレントヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、パージ可能ブロックのパージを繰り返す。

Cの関数► Block **MMChPurge*(long cbNeeded);
返り値はフリーブロックへのポインタ。

$A024　MMChMelt

引　数► long　cbNeeded　;必要な空きスペース

返り値► D0.L　= 0　エラー
≠ 0　フリーブロックへのポインタ

機　能► カレントヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、再配置可能ブロックの再配置とパージ可能ブロックのパージを繰り返す。

Cの関数► Block **MMChMelt*(long cbNeeded);
返り値はフリーブロックへのポインタ。

$A025　MMChReserve

引　数► long　cbNeeded　；必要な空きスペース
返り値► D0.L　= 0　エラー
　≠ 0　フリーブロックへのポインタ
機　能► カレントヒープゾーンに対して、cbNeeded で指定したサイズのフリーブロックが作成できるまで、再配置可能ブロックの再配置とパージ可能ブロックのパージを繰り返す。
Cの関数► Block *MMChReserve(long cbNeeded);
返り値はフリーブロックへのポインタ。

$A026　MMChFreeSize

引　数► なし
返り値► D0.L　カレントヒープゾーンのフリーブロックのサイズの総和
　= -1　エラー
機　能► カレントヒープのフリーブロックのサイズの総和を返す。
Cの関数► long MMChFreeSize(void);
返り値はフリーブロックのサイズの総和。

$A027　MMChGrowHeapGet

引　数► なし
返り値► D0.L　グローヒープゾーン関数のアドレス
機　能► カレントヒープのグローヒープゾーン関数のアドレスを返す。
Cの関数► int (*MMChGrowHeapGet(void))(Heap *hz, long cbNeeded);
返り値は関数のアドレス。

$A028　MMChGrowHeapSet

引　数► long　pGrowZone　グローヒープゾーン関数のアドレス
返り値► D0.L　前のグローヒープゾーン関数のアドレス
機　能► カレントヒープのグローヒープゾーン関数のアドレスを設定する。
Cの関数► int (*MMChGrowHeapSet(int (*pGrowZone)(Heap *hz,
long cbNeeded))) (Heap *hz, long cbNeeded);
返り値は前の関数のアドレス。

$A029　MMChPurgeGet

引　数► なし
返り値► D0.L　パージ関数のアドレス
機　能► カレントヒープゾーンのパージ関数のアドレスを返す。
Cの関数► void (**MMChPurgeGet*(void))(Heap *hz, _Handle h);
返り値は関数のアドレス。

$A02A　MMChPurgeSet

引　数► long　pPurge　；パージ関数のアドレス
返り値► D0.L　前のパージ関数のアドレス
機　能► カレントヒープゾーンのパージ関数のアドレスを設定する。
Cの関数► void(**MMChPurgeSet*(void (*pPurge)(Heap *hz, _Handle h)))(Heap *hz, _Handle h);
返り値は前の関数のアドレス。

$A02B　MMChCompactGet

引　数► なし
返り値► D0.L　再配置関数のアドレス
機　能► カレントヒープゾーンの再配置関数のアドレスを返す。
Cの関数► void (**MMChCompactGet*(void))(void);
返り値は関数のアドレス。

$A02C　MMChCompactSet

引　数► long　pReloc　；再配置関数のアドレス
返り値► D0.L　前の再配置関数のアドレス
機　能► カレントヒープゾーンの再配置関数のアドレスを設定する。
Cの関数► void (**MMChCompactSet*(void (*pReloc)(void)))(void);
返り値は前の関数のアドレス。

$A02D　MMPtrNew

引　数► long　hz　；ヒープゾーンのアドレス
long　logicalSize　；作成するブロックの論理サイズ
返り値► D0.L　= 0　エラー
≠ 0　作成されたブロックへのポインタ

機　能► hz で指定したヒープゾーンのなかに logicalSize の論理サイズを持つ再配置不能ブロックを、カレントヒープゾーン中に作成する。
再配置が発生する。

Cの関数► Pointer *MMPtrNew*(Heap *hz, long logicalSize);
返り値はブロックへのポインタ。

$A02E　MMPtrHeap

引　数► long　p　；所属するゾーンを調べるブロックへのポインタ

返り値► D0.L　= 0　エラー
≠ 0　ブロックが所属するヒープゾーンの先頭アドレス

機　能► p で指定した再配置不能ブロックが所属しているヒープゾーンの先頭アドレスを返す。

Cの関数► Heap **MMPtrHeap*(Pointer p);
返り値はヒープゾーンの先頭アドレス。

$A02F　MMPtrDispose

引　数► long　p　；廃棄するブロックへのポインタ

返り値► D0.L　= 0　正常終了
≠ 0　エラー

機　能► p で指定した再配置不能ブロックを廃棄する。

Cの関数► void *MMPtrDispose*(Pointer p);
返り値はない。

$A030　MMPtrSizeGet

引　数► long　p　；サイズを得るブロックへのポインタ

返り値► D0.L　論理サイズ
= -1　エラー

機　能► p で指定した再配置不能ブロックの論理サイズを返す。

Cの関数► long *MMPtrSizeGet*(Pointer p);
返り値は論理サイズ。

$A031　MMPtrSizeSet

引　数► long　p　；サイズを変更するブロックへのポインタ
long　newSize　；新しい論理サイズ

返り値► D0.L　= 0　正常終了
≠ 0　エラー

機　能► p で指定した再配置不能ブロックの論理サイズを newSize に変更する。
再配置が発生する。

Cの関数► int *MMPtrSizeSet*(Pointer p, long newSize);
返り値はエラーの有無。

$A032　MMPtrPropGet

引　数► long　p　; 属性を得るブロックへのポインタ

返り値► D0.L　ブロックの属性
= -1　エラー

機　能► p で指定した再配置不能ブロックの属性を得る。

Cの関数► int *MMPtrPropGet*(Pointer p);
返り値はブロックの属性。

$A033　MMPtrPropSet

引　数► long　p　; 属性を設定するブロックへのポインタ
long　flags　; 属性

返り値► D0.L　ブロックの属性
= -1　エラー

機　能► p で指定した再配置不能ブロックの属性として flags をセットする。

Cの関数► int *MMPtrPropSet*(Pointer p, int flags);
返り値はブロックの属性。

$A034　MMMstAllocate

引　数► long　hz　; ヒープゾーンのアドレス

返り値► D0.L　= 0　正常終了
≠ 0　エラー

機　能► hz で指定したヒープゾーンにマスタポインタブロックを追加作成する。
再配置が発生する。

Cの関数► int *MMMstAllocate*(Heap *hz);
返り値はエラーの有無。

$A035　MMMstBind

引　数► long　hz　; ヒープゾーンのアドレス

返り値► D0.L　= 0　エラー
　　　　　≠ 0　新しいハンドル
機　能► hz で指定したヒープゾーン中の、新しい空のハンドルを返す。メモリの確保などは行わない。
再配置が発生する。
Cの関数► Master *MMMstBind(Heap *hz);
返り値は新しいハンドル。

$A036　MMHdlNew

引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　logicalSize　；作成するブロックの論理サイズ
返り値► D0.L　= 0　エラー
　　　　　≠ 0　作成されたブロックへのハンドル
機　能► hz で指定したヒープゾーン中に、logicalSize の論理サイズを持つ再配置可能ブロックを作成する。
再配置が発生する。
Cの関数► _Handle *MMHdlNew*(Heap *hz, long logicalSize);
返り値はブロックへのハンドル。

$A037　MMHdlHeap

引　数► long　h　；所属するゾーンを調べるブロックへのハンドル
返り値► D0.L　= 0　エラー
　　　　　≠ 0　ブロックが所属するヒープゾーンの先頭アドレス
機　能► h で指定した再配置可能ブロックが所属しているヒープゾーンの先頭アドレスを返す。
Cの関数► Heap **MMHdlHeap*(_Handle h);
ヒープゾーンの先頭アドレス。

$A038　MMHdlDispose

引　数► long　h　；廃棄するブロックへのハンドル
返り値► D0.L　= 0　正常終了
　　　　　≠ 0　エラー
機　能► h で指定した再配置可能ブロックを廃棄する。
Cの関数► void *MMHdlDispose*(_Handle h);
返り値はない。

$A039　MMHdlSizeGet

引　数► long　h　; サイズを得るブロックへのハンドル
返り値► D0.L　論理サイズ
= -1　エラー
機　能► h で指定した再配置可能ブロックの論理サイズを返す。
Cの関数► long *MMHdlSizeGet*(_Handle h);
返り値は論理サイズ。

$A03A　MMHdlSizeSet

引　数► long　h　; サイズを変更するブロックへのハンドル
long　newSize　; 新しい論理サイズ
返り値► D0.L　= 0　正常終了
≠ 0　エラー
機　能► h で指定した再配置可能ブロックの論理サイズを newSize に変更する。
再配置が発生する。
Cの関数► int *MMHdlSizeSet*(_Handle h, long newSize);
返り値はエラーの有無。

$A03B　MMHdlEmpty

引　数► long　h　; 空にするハンドル
返り値► D0.L　= 0　正常終了
≠ 0　エラー
機　能► h で指定したハンドルを空にする。すなわち、h が指している再配置可能ブロックを廃棄し、空のハンドルだけを残す。
Cの関数► int *MMHdlEmpty*(_Handle h);
返り値はエラーの有無。

$A03C　MMHdlRealloc

引　数► long　h　; ハンドル
long　logicalByte　; 新しく確保するブロックの論理サイズ
返り値► D0.L　= 0　正常終了
≠ 0　エラー
機　能► ハンドル h に、logicalByte のサイズを持つ新しいブロックを割り当てる。
h が空のハンドルでなかった場合、h が指している再配置可能ブロックは廃棄され、新しく確保されたブロックが割り付けられる。

再配置が発生する。

Cの関数► int *MMHdlRealloc*(_Handle h, long logicalByte)

返り値はエラーの有無。

$A03D　MMHdlMoveHi

引　数►	long	h		;移動するブロックへのハンドル
返り値►	D0.L	= 0	正常終了	
		≠ 0	エラー	

機　能► ハンドルhが示すブロックを、できるだけ上位に移動させる。

Cの関数► int *MMHdlMoveHi*(_Handle h);

返り値はエラーの有無。

$A03E　MMHdlPropGet

引　数►	long	h		;属性を得るブロックへのハンドル
返り値►	D0.L	ブロックの属性		
		= -1	エラー	

機　能► hで指定した再配置可能ブロックの属性を得る。

Cの関数► int *MMHdlPropGet*(_Handle h);

返り値はブロックの属性。

$A03F　MMHdlPropSet

引　数►	long	h		;属性を設定するブロックへのハンドル
	long	flags		;属性
返り値►	D0.L	前の属性		
		= -1	エラー	

機　能► hで指定した再配置可能ブロックの属性をflagsに設定する。

Cの関数► int *MMHdlPropSet*(_Handle h, int flags);

返り値は前の属性。

$A040　MMHdlLock

引　数►	long	h		;ロックするブロックへのハンドル
返り値►	D0.L	= 0	正常終了	
		≠ 0	エラー	

機　能► hで指定した再配置可能ブロックをロックする。

Cの関数► int *MMHdlLock*(_Handle h);
返り値はエラーの有無。

$A041　MMHdlUnlock

引　数► long　h　; ロックを解除するブロックへのハンドル
返り値► D0.L　= 0　正常終了
　　　　　　≠ 0　エラー
機　能► h で指定した再配置可能ブロックのロックを解除する。
Cの関数► int *MMHdlUnlock*(_Handle h);
返り値はエラーの有無。

$A042　MMHdlPurge

引　数► long　h　; パージ可能に設定するブロックへのハンドル
返り値► D0.L　= 0　正常終了
　　　　　　≠ 0　エラー
機　能► h で指定した再配置可能ブロックをパージ可能に設定する。
Cの関数► int *MMHdlPurge*(_Handle h);
返り値はエラーの有無。

$A043　MMHdlNoPurge

引　数► long　h　; パージ不能に設定するブロックへのハンドル
返り値► D0.L　= 0　正常終了
　　　　　　≠ 0　エラー
機　能► h で指定した再配置可能ブロックをパージ不能に設定する。
Cの関数► int *MMHdlNoPurge*(_Handle h);
返り値はエラーの有無。

$A044　MMHdlResource

引　数► long　h　; リソースビットを ON にするブロックへのハンドル
返り値► D0.L　= 0　正常終了
　　　　　　≠ 0　エラー
機　能► **使用禁止コール。**
h で指定した再配置可能ブロックのリソースビットを ON にする。

$A045　　MMHdlNoResource

引　数► long　h　；リソースビットを OFF にするブロックへのハンドル

返り値► D0.L　= 0　正常終了
　　　　　　　≠ 0　エラー

機　能► **使用禁止コール。**
h で指定した再配置可能ブロックのリソースビットを OFF にする。

$A046　　MMHdlIns

引　数► long　h　；操作するブロックのハンドル
　　　　long　offset　；論理オフセット
　　　　long　length　；拡張するバイト数

返り値► D0.L　= 0　正常終了
　　　　　　　≠ 0　エラー

機　能► h で指定したブロックの offset からブロック終端までの内容を、length バイト上位方向に転送する。その際、ブロックは length バイト拡張される。再配置が発生する。

Cの関数► int *MMHdlIns*(_Handle h, long offset, long length);
返り値はエラーの有無。

$A047　　MMHdlDel

引　数► long　h　；操作するブロックのハンドル
　　　　long　offset　；論理オフセット
　　　　long　length　；削除するバイト数

返り値► D0.L　= 0　正常終了
　　　　　　　≠ 0　エラー

機　能► h で指定したブロックの offset からブロック終端までの内容を、length バイト下位方向に転送する。その際、ブロックは length バイト縮小される。再配置が発生する。

Cの関数► int *MMHdlDel*(_Handle h, long offset, long length);
返り値はエラーの有無。

$A048　　MMBlockUsrFlagGet

引　数► long　p　；ブロックヘッダのアドレス

返り値► D0.L　ユーザフラグの内容 (word)

機　能► **使用禁止コール。**
p で指定したブロックのユーザフラグの内容を返す。ブロックヘッダのアドレスであることに注意。

$A049　MMBlockUsrFlagSet

引　数► long　p　; ブロックヘッダのアドレス
long　usrflag　; ユーザフラグに設定する値
返り値► D0.L　前のユーザフラグ
機　能► **使用禁止コール。**
p で指定したブロックのユーザフラグの内容を usrflag にする。ブロックヘッダのアドレスであることに注意。

$A04A　MMBlockUsrWordGet

引　数► long　p　; ブロックヘッダのアドレス
返り値► D0.L　ユーザワードの内容 (long)
機　能► **使用禁止コール。**
p で指定したブロックのユーザワードの内容を返す。ブロックヘッダのアドレスであることに注意。

$A04B　MMBlockUsrWordSet

引　数► long　p　; ブロックヘッダのアドレス
long　usrword　; ユーザワードに設定する値
返り値► D0.L　前のユーザワードの内容
機　能► **使用禁止コール。**
p で指定するブロックのユーザワードに値を設定する。ブロックヘッダのアドレスであることに注意。

$A04C　MMMemAmiTPeach

引　数► long　hz　; ヒープゾーンのアドレス
long　rProc　; 再配置可能ブロックに対する処理
long　urProc　; 再配置不能ブロックに対する処理
返り値► なし
機　能► **使用禁止コール。**
hz で指定したヒープゾーン中に存在するブロックについて、再配置可能ブロックに対しては rProc で指定される関数を、再配置不能ブロックに対しては urProc

で指定される関数を実行する。
rProc，urProc の仕様は次のとおり。
rProc
引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　h　；再配置可能ブロックへのハンドル
urProc
引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　p　；再配置不能ブロックへのポインタ
どちらも返り値に意味を持たせることはできない。特にレジスタを保存する必要はない。

$A04D　MMMemHiReserve

引　数► long　hz　；ヒープゾーンのアドレス
　　　　long　cbNeeded　；必要な空きスペース
返り値► D0.L　= 0　エラー
　　　　　　≠ 0　フリーブロックへのポインタ
機　能► $A013 MMMemReserve ができるだけ下位にフリーブロックを作るのに対し、このコールはできるだけ上位にフリーブロックを作成する。
再配置が発生する。
Cの関数► Block *_MMMemHiReserve_(Heap *hz, long cbNeeded);
返り値はフリーブロックへのポインタ。

$A04E　MMPtrBlock

引　数► long　p　；ブロックヘッダを求めるブロックへのポインタ
返り値► D0.L　ブロックヘッダ/リザルトコード
機　能► p で指定したブロックのブロックヘッダのアドレスを返す。
Cの関数► Block *_MMPtrBlock_(Pointer p);
返り値はブロックへのポインタ。

$A04F　MMHdlBlock

引　数► long　h　；ブロックヘッダを求めるブロックへのハンドル
返り値► D0.L　ブロックヘッダ/リザルトコード
機　能► h で指定したブロックのブロックヘッダのアドレスを返す。
Cの関数► Block *_MMHdlBlock_(_Handle h);

返り値はブロックへのポインタ。

$A050　MMHdlMstGet

引　数► long　hz　;ヒープゾーンのアドレス
　　　　long　p　;ブロックへのポインタ
返り値► D0.L　マスタポインタ/リザルトコード
機　能► hz で指定したヒープゾーンのなかの、p で指定したブロックのマスタポインタのアドレスを返す。
Cの関数► Master **MMHdlMstGet*(Heap *hz, void *p);
返り値はマスタポインタ。

$A051　MMChHiReserve

引　数► long　cbNeeded　;必要な空きスペース
返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　フリーブロックへのポインタ
機　能► カレントヒープゾーンに対して、$A04D MMMemHiReserve と同様、できるだけ上位にフリーブロックを作成する。
再配置が発生する。
Cの関数► Block **MMChHiReserve*(long cbNeeded);
返り値はフリーブロックへのポインタ。

$A052　MMChUsrFlagGet

引　数► なし
返り値► D0.L　カレントヒープゾーンのユーザフラグの内容
機　能► **使用禁止コール。**
カレントヒープゾーンの、ユーザフラグの内容を返す。

$A053　MMChUsrFlagSet

引　数► long　usrflag　;カレントヒープゾーンのユーザフラグに設定する値
返り値► D0.L　前のユーザフラグの内容 (word)
機　能► **使用禁止コール。**
カレントヒープゾーンのユーザフラグの内容を usrflag に設定する。

$A054　　MMChUsrWordGet

引　数► なし

返り値► D0.L　　カレントヒープゾーンのユーザワードの内容 (long)

機　能► **使用禁止コール。**

カレントヒープゾーンのユーザワードの内容を返す。

$A055　　MMChUsrWordSet

引　数► long　　usrword　　　; カレントヒープゾーンのユーザワードに設定する値

返り値► D0.L　　前のユーザワードの内容

機　能► **使用禁止コール。**

カレントヒープゾーンのユーザワードの内容を usrflag に設定する。

エクセプションマン

#include <CONSOLE.H>

$A068　EXEnVDISPST

引　数► long　usrEntry　；割り込みルーチンのアドレス

返り値► D0.L　リザルトコード

機　能► V-DISP 割り込みに、usrEntry で指定された割り込みルーチンを設定する。

Cの関数► int *EXEnVDISPST*(void (*usrEntry)(void));

返り値はリザルトコード。

$A069　EXDeVDISPST

引　数► long　usrEntry　；割り込みルーチンのアドレス

返り値► D0.L　リザルトコード

機　能► V-DISP に登録されている割り込みルーチンのうち、usrEntry で指定された割り込みルーチンを削除する。

Cの関数► int *EXDeVDISPST*(void (*usrEntry)(void));

返り値はリザルトコード。

マウスマン

#include <CONSOLE.H>

$A06A　MSInitCsr

引　数► なし
返り値► なし
機　能► マウスカーソルを初期化する。
カーソルパターンは標準のものに、カーソルレベルは 0 になる。
Cの関数► void *MSInitCsr*(void);
返り値はない。

$A06B　MSShowCsr

引　数► なし
返り値► なし
機　能► カーソルレベルを +1 する。
Cの関数► void *MSShowCsr*(void);
返り値はない。

$A06C　MSHideCsr

引　数► なし
返り値► なし
機　能► カーソルレベルを −1 する。
Cの関数► void *MSHideCsr*(void);
返り値はない。

$A06D　MSSetCsr

引　数► long　TXcsrHdl　;カーソルレコードへのハンドル
返り値► なし
機　能► カーソルパターンを変更する。カーソルレベルは変化しない。
疑似ハンドルも可。
Cの関数► void *MSSetCsr*(MsCsr **TXcsrHdl);
返り値はない。

$A06E　MSObscureCsr

引　数► なし
返り値► なし
機　能► マウスカーソルの表示をやめる。カーソルレベルは変化しない。
カーソルレベルが 0 の場合、マウスが移動するか、$A06D MSSetCsr を呼ぶことでマウスカーソルが再表示される。
Cの関数► void *MSObscureCsr*(void);
返り値はない。

$A06F　MSShieldCsr

引　数► long　rectPtr　;レクタングルレコードへのポインタ
long　ofsPt　;オフセットを示すポイント
返り値► なし
機　能► マウスカーソルが rectPtr で指定した矩形領域と重なるかどうかを調べ、それに応じた処理を行う。

- 重なる場合
  カーソルレベルを −1 して、マウスカーソルを消す。
- 重ならない場合
  カーソルレベルを −1 する。

マウスを移動するか、$A06D MSSetCsr を呼ぶことで実際にマウスカーソルが消される。
rectPtr で指定したレクタングルがローカル座標系で示されている場合、加算することでグローバル座標系に変換できるような ofsPt を指定する。グローバル座標系の場合は (0,0) を指定する。
Cの関数► void *MSShieldCsr*(Rect *rectPtr, LPoint ofsPt);
返り値はない。

$A070　MSGetCurMsr

引　数► なし
返り値► D0.L　マウスレコードへのポインタ
機　能► マウスレコードへのポインタを返す。
Cの関数► Mouse **MSGetCurMsr*(void);
返り値はマウスレコードへのポインタ。

$A071　MSMultiGet

引　数► なし
返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　マウススピードの値
機　能► マウススピードの値を返す。
（移動量 ＝ マウス変移 × マウススピード / 256）
Cの関数► long *MSMultiGet*(void);
返り値はマウススピードの値。

$A072　MSMultiSet

引　数► long　msSpeed　；マウススピード (1〜65535)
返り値► D0.L　= 0　エラー
　　　　　　　≠ 0　前のマウススピードの値
機　能► マウススピードの値を msSpeed にする。
（移動量 ＝ マウス変移 × マウススピード / 256）
Cの関数► long *MSMultiSet*(long msSpeed);
返り値は前のマウススピードの値。

$A076　MSBoundGet　2.0

引　数► なし
返り値► D0.L　レクタングルレコードへのポインタ
機　能► マウスの移動範囲を意味するレクタングルレコード（グローバル座標系）へのポインタを返す。

$A077　MSBoundSet　2.0

引　数► long　rectPtr　；レクタングルレコードへのポインタ
返り値► D0.L　レクタングルレコードへのポインタ
機　能► **使用禁止コール。**
マウスの移動範囲 rectPtr で示される範囲に設定する。

$A078　MSMove　2.0

引　数► long　pt　；マウスポインタのグローバル座標
返り値► D0.L　レクタングルレコードへのポインタ

機　能►　**使用禁止コール。**

マウスポインタの位置を pt に移動する。

アニメーションマン

#include <CONSOLE.H>

$A073　EXAnimStart

引　数► long　nmbPtn　; パターン数
　　　　long　vcount　; 書き換えカウンタ初期値 (1〜)
　　　　long　ptnList　; カーソルレコードへのハンドルの配列

返り値► D0.L　リザルトコード

機　能► カーソルのアニメーションを開始する。

Cの関数► int *EXAnimStart*(int nmbPtn,int vcount,MsCsr ***ptnList);
　　　　返り値はリザルトコード。

$A074　EXAnimEnd

引　数► なし

返り値► D0.L　リザルトコード

機　能► カーソルのアニメーションを終了する。

Cの関数► int *EXAnimEnd*(void);
　　　　返り値はリザルトコード。

$A075　EXAnimTest

引　数► なし

返り値► D0.L　= 0　アニメーションしていない
　　　　　　　= -1　アニメーション実行中

機　能► カーソルのアニメーション状態を調べる。

Cの関数► BOOLEAN *EXAnimTest*(void);
　　　　返り値はアニメーション状態。

キーボードマン

#include <CONSOLE.H>

$A086　KBMapGet

引　数► long　kbRec　；キーボードレコードのアドレス
返り値► D0.L　キーマップのアドレス
機　能► kbRecで指定したキーボードレコードのなかの、キーマップのアドレスを返す。
Cの関数► char **KBMapGet*(KBoard *kbRec);
返り値はキーマップのアドレス。

$A087　KBShiftGet

引　数► long　kbRec　；キーボードレコードのアドレス
返り値► D0.L　シフトキービット
機　能► kbRecで指定したキーボードレコードのなかのシフトキービットを返す。
Cの関数► long *KBShiftGet*(KBoard *kbRec);
返り値はシフトキービット。

$A088　KBShiftSet

引　数► long　kbRec　；キーボードレコードのアドレス
long　newSBit　；新しいシフトキービットの値
返り値► D0.L　前のシフトキービット
機　能► kbRecで指定したキーボードレコードのなかのシフトキービットをnewSBitの値に設定する。
Cの関数► long *KBShiftSet*(KBoard *kbRec, long newSBit);
返り値は前のシフトキービット。

$A089　KBSimulate

引　数► long　kbRec　；キーボードレコードのアドレス
long　kData　；キーデータ
返り値► D0.L　リザルトコード
機　能► キーボードからkDataで示される1文字を入力したふりをする。
kDataの値は、キーボードから送られてくるデータと同様。つまり、キーコー

ド（≠ JIS コード）に‘‘押された（0x00）/離された（0x80）’’情報を加えたもの。

Cの関数► int *KBSimulate*(KBoard *kbRec, long kData);
返り値はリザルトコード。

$A08A　KBScan

引　数► long　kbRec　；キーボードレコードのアドレス

返り値► D0.L　キーバッファ先頭のデータ

機　能► kbRec で指定したキーボードレコードのなかのキーバッファの先頭のデータを参照する。キーバッファが空の場合は 0 が返る。

Cの関数► int *KBScan*(KBoard *kbRec);
返り値はキーバッファ先頭のデータ。

$A08B　KBGet

引　数► long　kbRec　；キーボードレコードのアドレス

返り値► D0.L　キーバッファ先頭のデータ

機　能► kbRec で指定したキーボードレコードのなかのキーバッファの先頭のデータを得る。得られたデータはバッファから除かれる。キーバッファが空の場合は 0 が返る。

Cの関数► int *KBGet*(KBoard *kbRec);
返り値はキーバッファ先頭のデータ。

$A08C　KBEmpty

引　数► long　kbRec　；キーボードレコードのアドレス

返り値► なし

機　能► kbRec で指定したキーボードレコードのなかのキーバッファを空にする。

Cの関数► void *KBEmpty*(KBoard *kbRec);
返り値はない。

$A08D　KBInit

引　数► long　buflen　；バッファのバイト数
long　flags　；フラグ

bit0	HALT
bit1	RESET
bit2	OLD
bit3	LED
bit4	CLICK
bit5	ASSIGN
bit6	REPEAT

返り値► D0.L　= 0　エラー
　　　　　≠ 0　キーボードレコードのアドレス

機　能► キーボードマンを初期化する。

Cの関数► KBoard **KBInit*(int buflen, int flags);
返り値はキーボードレコードのアドレス。

$A08E　KBTini

引　数► long　kbRec　;キーボードレコードのアドレス

返り値► なし

機　能► キーボードマンの終了処理を行う。

Cの関数► void *KBTini*(KBoard kbRec);
返り値はない。

$A08F　KBCurKbrGet

引　数► なし

返り値► D0.L　キーボードレコードのアドレス

機　能► キーボードレコードのアドレスを返す。

Cの関数► KBoard **KBCurKbrGet*(void);
返り値はキーボードレコードのアドレス。

$A090　KBOldOnGet

引　数► long　kbRec　;キーボードレコードのアドレス

返り値► D0.L　= 0　OLD OFF
　　　　　≠ 0　OLD ON

機　能► kbRecで指定した、キーボードレコードに設定されているOLD ONフラグの設定状況を返す。

Cの関数► BOOLEAN *KBOldOnGet*(KBoard *kbRec);
返り値はOLD ONフラグの状態。

$A091 KBOldOnSet

引　数► long　kbRec　; キーボードレコードのアドレス
long　newOld　; 新しい OLD の設定

0	OLD OFF
-1	OLD ON

返り値► D0.L　前の OLD の設定状況

機　能► kbRec で指定したキーボードレコードに、OLD の設定を行う。

Cの関数► BOOLEAN *KBOldOnSet*(KBoard *kbRec, BOOLEAN newOld);
返り値は前の OLD ON フラグの状態。

$A092 KBFlagGet

引　数► long　kbRec　; キーボードレコードのアドレス

返り値► D0.L　キーボードマンのフラグ

bit0	Halt
bit1	ResetOn
bit2	OldOn
bit3	LedOn
bit4	ClickOn
bit5	RepeatOn
bit6	AssignOn

機　能► キーボードマンのフラグ類を一括して返す。

Cの関数► int *KBFlagGet*(KBoard *kbRec);
返り値はフラグの状態。

$A093 KBFlagSet

引　数► long　kbRec　; キーボードレコードのアドレス
long　flags　; フラグ類の状態

bit0	Halt
bit1	ResetOn
bit2	OldOn
bit3	LedOn
bit4	ClickOn
bit5	RepeatOn
bit6	AssignOn

返り値► D0.L　前のフラグの状態

機　能► キーボードマンのフラグ類を一括して設定する。

Cの関数► int *KBFlagSet*(KBoard *kbRec, int flags);
返り値は前のフラグの状態。

キーマン

#include <CONSOLE.H>

$A09A　KMEmpty

引　数► long　kmRecPtr　；キーマンレコードへのポインタ
返り値► なし
機　能► kmRecPtr で指定したキーマンレコードのなかのメッセージキューを初期化し、空にする。
Cの関数► void *KMEmpty*(Key *kmRecPtr);
返り値はない。

$A09B　KMPost

引　数► long　kmRecPtr　；キーマンレコードへのポインタ
long　kData　；キーデータ（最下位 8 ビットのみ有効）
long　ascCode　；ASCII コード（最下位 8 ビットのみ有効）
返り値► D0.L　リザルトコード
機　能► kData と ascCode からメッセージを作り、kmRecPtr で指定したキーマンレコードのなかのメッセージキューに登録する。
Cの関数► int *KMPost*(Key *kmRecPtr, int kData, int ascCode);
返り値はリザルトコード。

$A09C　KMAscJobSet

引　数► long　kmRecPtr　；キーマンレコードへのポインタ
long　jobAdr　；処理ルーチンのアドレス
返り値► D0.L　前の処理ルーチンのアドレス
機　能► Ascjob として jobAdr で指定したルーチンを登録する。
Cの関数► int (**KMAscJobSet*(Key *kmRecPtr, int(*jobAdr)(void)))(void);
返り値は前の処理ルーチンのアドレス。

$A09D　KMSimulate

引　数► long　kmRecPtr　；キーマンレコードへのポインタ

long　kData　; キーデータ
返り値► D0.L　リザルトコード
機　能► kData で指定したキーデータによってメッセージを生成する。シフトキービット、マップの操作も行う。
Cの関数► int *KMSimulate*(Key *kmRecPtr, int kData);
返り値はリザルトコード。

$A09E　KMTask

引　数► long　kmRecPtr　; キーマンレコードへのポインタ
返り値► D0.L　なし
機　能► キーマンを駆動する。
イベントマンが使用する。
Cの関数► void *KMTask*(Key *kmRecPtr);
返り値はない。

$A09F　KMInit

引　数► long　buflen　; メッセージキューのバイト数
返り値► D0.L　= 0　エラー
≠ 0　キーマンレコードへのポインタ
機　能► キーマンを初期化する。キーボードマンが初期化されている必要がある。
Cの関数► Key **KMInit*(long buflen);
返り値はキーマンレコードへのポインタ。

$A0A0　KMTini

引　数► long　kmRecPtr　; キーマンレコードへのポインタ
返り値► なし
機　能► キーマンの終了処理を行う。
Cの関数► void *KMTini*(Key *kmRecPtr);
返り値はない。

$A0A1　KMCurKmrGet

引　数► なし
返り値► D0.L　キーマンレコードへのポインタ
機　能► キーマンレコードへのポインタを返す。

キーマン

Cの関数 ► Key *KMCurKmrGet(void);

返り値はキーマンレコードへのポインタ。

イベントマン

#include <EVENT.H>

$A0A2　　EMInit

引　数► なし
返り値► なし
機　能► イベントマンを初期化する。
イベントキュー、割り込みルーチンが初期化される。
Cの関数► void *EMInit*(void);
返り値はない。

$A0A3　　EMTini

引　数► なし
返り値► D0.L　　リザルトコード
機　能► イベントマンの終了処理を行う。
割り込みルーチンを開放し、ベクタを元に戻す。
Cの関数► void *EMTini*(void);
返り値はない。

$A0A4　　EMSet

引　数► long　what　　；発生させるイベントのイベントコード
long　whom　　；イベントに付随する情報
返り値► D0.L　　リザルトコード
機　能► what で指定したイベントをイベントキューに登録する。
イベントコードとそれに付随する情報の内容は以下のとおり。

イベントコード	イベントに付随する情報
0 : E_IDLE	E_IDLE イベントを発生させることはできない
1 : E_MSLDOWN	0
2 : E_MSLUP	0
3 : E_MSRDOWN	0
4 : E_MSRUP	0
5 : E_KEYDOWN	上位ワード: キーコード、下位ワード:ASCII コード
6 : E_KEYUP	上位ワード: キーコード、下位ワード:ASCII コード
7 : E_UPDATE	ウィンドウレコードへのポインタ
9 : E_ACTIVATE	ウィンドウレコードへのポインタ
12 : E_SYSTEM1	イベントマンの管理外(タスクマンが管理する)
13 : E_SYSTEM2	イベントマンの管理外(タスクマンが管理する)
14 : E_SYSTEM3	イベントマンの管理外(タスクマンが管理する)
15 : E_SYSTEM4	イベントマンの管理外(タスクマンが管理する)

Cの関数► int *EMSet*(int what, long whom);
返り値はリザルトコード。

$A0A5　　EMGet

引　数► word　mask　;イベントマスク
　　　　long　eventRecPtr　;イベントレコードへのポインタ

返り値► D0.L　= 0　イベントは得られなかった
　　　　　　　≠ 0　イベントを得ることができた

機　能► イベントキューから mask の各ビットで指定したイベントを取り出し、削除する。取り出したイベントの内容は eventRecPtr で指定したイベントレコードに返る。

Cの関数► BOOLEAN *EMGet*(int mask, Event *eventRecPtr);
返り値はイベント取り出しの成否。

$A0A6　　EMScan

引　数► word　mask　;イベントマスク
　　　　long　eventRecPtr　;イベントレコードへのポインタ

返り値► D0.L　= 0　イベントは発見できなかった
　　　　　　　≠ 0　イベントを発見できた

機　能► イベントキューから mask の各ビットで指定したイベントのうち、イベントキューのなかで最初に見つかったものを参照する。イベントの内容は eventRecPtr で指定したイベントレコードに返る。

Cの関数► int *EMScan*(int mask, Event *eventRecPtr);
返り値はイベント参照の成否。

$A0A7　EMMSLoc

引　数► なし
返り値► D0.L　　マウス位置を示すポイント(ローカル座標)/リザルトコード
機　能► マウスの座標を返す。
あらかじめカレントグラフレコードをセットしておく必要がある。
Cの関数► LPoint *EMMSLoc*(void);
返り値はマウスのローカル座標を意味するポイント。

$A0A8　EMLBttn

引　数► なし
返り値► D0.L　　= 0　　押されていない
≠ 0　　押されている
機　能► マウスの左ボタンの状態を返す。
Cの関数► BOOLEAN *EMLBttn*(void);
返り値は左ボタンの状態。

$A0A9　EMRBttn

引　数► なし
返り値► D0.L　　= 0　　押されていない
≠ 0　　押されている
機　能► マウスの右ボタンの状態を返す。
Cの関数► BOOLEAN *EMRBttn*(void);
返り値は右ボタンの状態。

$A0AA　EMLStill

引　数► なし
返り値► D0.L　　= 0　　押されたままではない
≠ 0　　押されたまま
機　能► マウスの左ボタンが押されたままかどうかを調べる。
Cの関数► BOOLEAN *EMLStill*(void);
返り値は左ボタンの状態。

$A0AB　EMRStill

引　数► なし

返り値► D0.L　　= 0　　押されたままではない
　　　　　　　≠ 0　　押されたまま
機　能► マウスの右ボタンが押されたままかどうかを調べる。
Cの関数► BOOLEAN *EMRStill*(void);
　　　　返り値は右ボタンの状態。

$A0AC　　EMLWait

引　数► なし
返り値► D0.L　　= 0　　押されたままではない
　　　　　　　≠ 0　　押されたまま
機　能► マウスの左ボタンが押されたままかどうかを調べ、離されていれば離された時点で登録されたE_MSLUPイベントを取り除く。
Cの関数► BOOLEAN *EMLWait*(void);
　　　　返り値は左ボタンの状態。

$A0AD　　EMRWait

引　数► なし
返り値► D0.L　　= 0　　押されたままではない
　　　　　　　≠ 0　　押されたまま
機　能► マウスの右ボタンが押されたままかどうかを調べ、離されていれば離された時点で登録されたE_MSRUPイベントを取り除く。
Cの関数► BOOLEAN *EMRWait*(void);
　　　　返り値は右ボタンの状態。

$A0AE　　EMKMapGet

引　数► long　　kmapBuf　　; キーマップが返るバッファのアドレス
返り値► D0.L　　リザルトコード
機　能► kmapBufで指定された128バイトのバッファに、キーマップの内容をコピーする。
Cの関数► void *EMKMapGet*(char **kmapBuf);
　　　　返り値はない。

$A0AF　　EMSysTime

引　数► なし
返り値► D0.L　　システム時間

機　能► 現在のシステム時間（1/100 秒単位）を返す。
Cの関数► unsigned long *EMSysTime*(void);
返り値はシステム時間。

$A0B0　EMDClickGet

引　数► なし
返り値► D0.L　ダブルクリック基準時間（1/100 秒単位）
機　能► 設定されているダブルクリック基準時間を返す。
Cの関数► unsigned long *EMDClickGet*(void);
返り値はダブルクリック基準時間。

$A0B1　EMBlinkGet

引　数► なし
返り値► D0.L　カーソル点滅基準時間（1/100 秒単位）
機　能► 設定されているカーソル点滅基準時間を返す。
Cの関数► unsigned long *EMBlinkGet*(void);
返り値はカーソル点滅基準時間。

$A0B2　EMClean

引　数► word　mask　；削除するイベントを示すイベントマスク
word　sMask　；削除を終了するイベントを示すイベントマスク
返り値► D0.L　リザルトコード
機　能► イベントキューのなかで mask の各ビットで指定したイベントを削除する。sMask の各ビットで指定したイベントを発見した時点で削除を終了する。
Cの関数► int *EMClean*(int mask, int sMask);
返り値はリザルトコード。

$A0B3　EMMaskSet

引　数► word　mask　；イベントマスク
返り値► なし
機　能► $A0A4 EMSet でイベントを登録する際に参照される、登録できるイベントを制限するためのマスクを mask で指定する。立っているビットに対応するイベントのみ登録可となる。

Cの関数► void *EMMaskSet*(int mask);
返り値はない。

$A0B4 EMDTTskSet

引　数► long TTskAdr ; ユーザ背景描画ルーチンのアドレス
返り値► なし
機　能► SX-WINDOW ver.3.0 以降からは使用できないコール。
背景書き換え時に呼び出されるユーザルーチンを登録する。 0 を指定すると何もしないようになる。

$A0B5 EMDClickSet

引　数► long clickTime ; ダブルクリック基準時間（1/100 単位）
返り値► なし
機　能► ダブルクリックの基準時間を clickTime に設定する。
Cの関数► void *EMDClickSet*(unsigned long clickTime);
返り値はない。

$A0B6 EMBlinkSet

引　数► long blinkTime ; カーソル点滅基準時間（1/100 単位）
返り値► なし
機　能► カーソル点滅の基準時間を blinkTime に設定する。
Cの関数► void *EMBlinkSet*(unsigned long blinkTime);
返り値はない。

$A0B7 EMEnCross

引　数► word d ; つねに 0
返り値► D0.L リザルトコード
機　能► マウスポインタを踏切カーソルにする。
Cの関数► int *EMEnCross*(void);
引数として 0 を与える必要はない。
返り値はリザルトコード。

$A0B8　　EMDeCross

引　数► なし
返り値► D0.L　　リザルトコード
機　能► マウスポインタを元に戻す。
Cの関数► int *EMDeCross*(void);
返り値はリザルトコード。

リソースマン

#include <RESOURCE.H>

$A0D9　　RMInit

引　数► なし
返り値► なし
機　能► リソースマンを初期化する。
SX シェル上で動作するプログラムは、これを呼び出してはならない。
Cの関数► int *RMInit*(void);
返り値は意味を持たない。

$A0DA　　RMTini

引　数► なし
返り値► なし
機　能► リソースマンの終了処理を行う。
SX シェル上で動作するプログラムは、これを呼び出してはならない。
Cの関数► void *RMTini*(void);
返り値はない。

$A0DB　　RMResNew

引　数► なし
返り値► D0.L　　リザルトコード
A0.L　　リソースマップへのハンドル
機　能► 新しいリソースマップをメモリ上に作成する。
実際にファイルが作成されるわけではない。
作成直後のリソースマップの内容は空である。
再配置が発生する。
Cの関数► Handle *RMResNew*(void);
返り値はリソースマップへのハンドル。
負の値の場合はリザルトコード。

$A0DC　　RMRscAdd

引　数► long　Type　　；追加するリソースのタイプ
word　ID　　；追加するリソースID
long　h　　；追加するリソースへのハンドル
long　Size　　；追加するリソースのサイズ

返り値► D0.L　リザルトコード
A0.L　新しいリソースへのハンドル

機　能► カレントのリソースマップにリソースを追加する。
リソースは、リソースマンが新しく確保したブロックにコピーされる。
疑似ハンドルも可。再配置が発生する。

Cの関数► _Handle *RMRscAdd*(long Type,int ID,_Handle h,long Size);
返り値は新しいリソースへのハンドル。
負の値の場合はリザルトコード。

$A0DD　　RMRscRemove

引　数► long　Type　　；削除するリソースのタイプ
word　ID　　；削除するリソースID

返り値► D0.L　リザルトコード

機　能► TypeとIDで指定したリソースを、カレントリソースマップから削除する。
再配置が発生する。

Cの関数► int *RMRscRemove*(long Type, int ID);
返り値はリザルトコード。

$A0DE　　RMTypeRemove

引　数► long　Type　　；削除するリソース群のタイプ

返り値► D0.L　リザルトコード

機　能► カレントリソースマップの、Typeで指定するタイプのリソース群をすべて削除する。
再配置が発生する。

Cの関数► int *RMTypeRemove*(long Type);
返り値はリザルトコード。

$A0DF　　RMResDispose

引　数► なし

返り値► D0.L　リザルトコード

機　能► カレントリソースマップをメモリから削除する。オープン中であった場合でもクローズしない。
再配置が発生する。

Cの関数► int *RMResDispose*(void);
返り値はリザルトコード。

$A0E0　RMResOpen

引　数► long　Name　;リソースファイル名

返り値► D0.L　リザルトコード
A0.L　リソースマップへのハンドル

機　能► Name で指定したリソースファイルをオープンして、カレントとする。この時点では、リソースマップのみが読み込まれ、ファイルはオープン状態のままである。
再配置が発生する。

Cの関数► Handle *RMResOpen*(const char *Name);
返り値はリソースマップへのハンドル。
負の値の場合はリザルトコード。

$A0E1　RMScGet

引　数► long　Type　;Get したいリソースのタイプ
word　ID　;Get したいリソース ID

返り値► D0.L　リザルトコード
A0.L　リソースへのハンドル

機　能► Type と ID で指定したリソースを探し、発見できた場合はハンドルを返す。
再配置が発生する。
SX-WINDOW ver.3.0 以降では、タスクマンがこのコールをフックして、ローカルリソース対応している。

Cの関数► _Handle *RMScGet*(long Type, int ID);
返り値はリソースへのハンドル。

$A0E2　RMResClose

引　数► long　Name　;ファイル名

返り値► D0.L　リザルトコード

機　能► カレントリソースマップとその下のリソース群を Name で指定したファイルにセーブして、メモリから削除する。カレントは次のリソースマップに移る。
再配置が発生する。

Cの関数► int *RMResClose*(const char *Name);
返り値はリザルトコード。

$A0E3　RMResRemove

引　数► なし
返り値► D0.L　リザルトコード
機　能► カレントリソースマップとその下のリソース群をメモリから削除する。オープン中の場合はクローズする。カレントは次のリソースマップに移る。
再配置が発生する。
Cの関数► int *RMResRemove*(void);
返り値はリザルトコード。

$A0E4　RMCurResSet

引　数► long　ResMap　;リソースマップのハンドル
返り値► A0.L　前のカレントリソースマップのハンドル
機　能► ResMapで指定したメモリ中のリソースマップをカレントにする。
Cの関数► Handle *RMCurResSet*(Handle ResMap);
返り値は前のカレントリソースマップのハンドル。

$A0E5　RMRscRelease

引　数► long　Rsc　;Releaseするリソースのハンドル
返り値► D0.L　リザルトコード
A0.L　ハンドルがそのまま返る（エラーの場合は0）
機　能► Rscで指定したリソースをリソースマンの管理から外し、メモリから削除する。
再配置が発生する。
Cの関数► int *RMRscRelease*(_Handle Rsc);
返り値はリザルトコード。

$A0E6　RMRscDetach

引　数► long　Rsc　;Detachするリソースのハンドル
返り値► D0.L　リザルトコード
A0.L　ハンドルがそのまま返る（エラーの場合は0）
機　能► Rscで指定したリソースをリソースマンの管理から外す。メモリからは削除しない。
再配置が発生する。

Cの関数► int *RMRscDetach*(_Handle Rsc);
返り値はリザルトコード。

$A0E7　RMMaxIDGet

引　数► long　Type　; 最大 ID を調べるタイプ
返り値► D0.L　Type の最大 ID/リザルトコード
機　能► Type で指定したリソース群のなかで最大の ID を返す。
Cの関数► int *RMMaxIDGet*(long Type);
返り値は Type の最大 ID/リザルトコード。

$A0E8　RMResSave

引　数► long　Name　; リソースファイル名
返り値► D0.L　リザルトコード
A0.L　リソースファイル名がそのまま返る
機　能► カレントリソースマップとその下のリソース群を Name で指定したファイルにセーブする。メモリから削除しない。カレントリソースマップは変更されない。再配置が発生する。
Cの関数► int *RMResSave*(const char *Name);
返り値はリザルトコード。

$A0E9　RMHdlToRsc

引　数► long　Rsc　; リソースマップを得るリソースへのハンドル
返り値► D0.L　リソースマップへのハンドル
機　能► リソースマン用。
Rsc で指定したリソースが所属するリソースマップのハンドルが返る。
Cの関数► Handle *RMHdlToRsc*(_Handle Rsc);
返り値はリソースマップへのハンドル。

$A0EA　RMCurResGet

引　数► なし
返り値► D0.L　カレントリソースマップへのハンドル
A0.L　カレントリソースマップへのハンドル
機　能► カレントリソースマップへのハンドルを返す。
Cの関数► Handle *RMCurResGet*(void);
返り値はカレントリソースマップへのハンドル。

$A0EB　　RMLastResGet

引　数► なし
返り値► D0.L　　最終リソースマップへのハンドル
　　　　A0.L　　最終リソースマップへのハンドル
機　能► 最終リソースマップ（最後にオープンされたリソースマップ）へのハンドルを返す。
Cの関数► Handle *RMLastResGet*(void);
　　　　返り値は最終リソースマップへのハンドル。

$A0EC　　RMResLoad

引　数► なし
返り値► D0.L　　リザルトコード
機　能► カレントリソースマップの下のリソースを、すべて読み込んでメモリに置く。
　　　　ファイルはクローズされる。
　　　　再配置が発生する。
Cの関数► int *RMResLoad*(void);
　　　　返り値はリザルトコード。

$A0ED　　RMResLinkGet

引　数► long　　ResMap　　　　；リソースマップへのハンドル
返り値► D0.L　　次のリソースマップへのハンドル
機　能► 指定したリソースマップの次のリソースマップを得る。
Cの関数► Handle *RMResLinkGet*(Handle ResMap);
　　　　返り値は次のリソースマップへのハンドル。

$A0EE　　RMResTypeList

引　数► long　　argc　　　　；タイプの数が返るバッファ（1ロングワード）のアドレス
　　　　long　　argv　　　　；タイプのリスト（タイプの数 × 1ロングワード）へのハンドルが返るバッファのアドレス
　　　　long　　ResMap　　　；リソースマップへのハンドル
返り値► D0.L　　= 0　　正常終了
　　　　　　　　≠ 0　　エラー
機　能► 指定したリソースマップに登録されているタイプの数とリストを得る。

リストは、タイプ名（1ロングワード）がタイプの数だけ並んでいる構造で、再配置可能ブロックとして作成される。末尾は0.L。**リストが不要になったら、廃棄する必要がある。**
再配置が発生する。

Cの関数► int *RMResTypeList*(int *argc,long ***argv,Handle ResMap);
返り値は結果を意味する数値。

$A0EF RMResIDList

引　数►

long	argc	; IDの数が返るバッファ（1ロングワード）のアドレス
long	argv	; IDのリスト（IDの数 × 1ワード）へのハンドルが返るバッファのアドレス
long	ResMap	; リソースマップへのハンドル
long	Type	; タイプ

返り値► D0.L　= 0　正常終了
　　　　　$\neq$ 0　エラー

機　能► 指定したリソースマップに登録されているタイプのIDの数とリストを得る。
リストはID（1ワード）がIDの数だけ並んでいる構造で、再配置可能ブロックとして作成される。末尾は0.W。**リストが不要になったら、廃棄する必要がある。**
再配置が発生する。

Cの関数► int *RMResIDList*(int *argc, short ***argv, Handle ResMap, long Type);
返り値は結果を意味する数値。

グラフマン

#include <SXGRAPH.H>

$A12D　GMOpenGraph

引　数► word　scrType　；初期化するグラフレコードのタイプ
long　graphPtr　；グラフレコードへのポインタ
返り値► D0.L　リザルトコード
A0.L　グラフレコードへのポインタが返る
機　能► GraphPtr で指定したグラフレコードを、scrType で指定したタイプのグラフレコードとして初期化し、カレントにする。
グラフレコードのメンバである bitmap のためのリージョンレコードを再配置不能ブロックとして作成する。
疑似ポインタも可。
再配置が発生する。
Cの関数► int *GMOpenGraph*(int scrType, Graph *graphPtr);
返り値はリザルトコード。

$A12E　GMCloseGraph

引　数► long　graphPtr　；グラフレコードへのポインタ
返り値► D0.L　リザルトコード
A0.L　グラフレコードへのポインタが返る
機　能► graphPtr で指定したグラフレコードをクローズする。
グラフレコードのメンバであった各種リージョンレコードが占めていたブロックは廃棄される。
疑似ポインタも可。
再配置が発生する。
Cの関数► int *GMCloseGraph*(Graph *graphPtr);
返り値はリザルトコード。

$A130　GMInitGraph

引　数► long　graphPtr　；グラフレコードへのポインタ
返り値► D0.L　リザルトコード
A0.L　グラフレコードへのポインタが返る
機　能► GraphPtr で指定したグラフレコードを初期化する。

グラフレコードのメンバであるビジブルリージョン、クリップリージョンのためのリージョンレコードを再配置可能ブロックとして作成する。$A12D GMOpenGraph を実行したあとに使用すること。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMInitGraph*(Graph *graphPtr);
返り値はリザルトコード。

$A131　GMSetGraph

引　数► long　graphPtr　;グラフレコードへのポインタ
返り値► D0.L　リザルトコード
A0.L　グラフレコードへのポインタ
機　能► graphPtr で指定したグラフレコードをカレントにする。
疑似ポインタも可。
Cの関数► int *GMSetGraph*(Graph *graphPtr);
返り値はリザルトコード。

$A132　GMGetGraph

引　数► なし
返り値► D0.L　= 0
A0.L　カレントグラフレコードへのポインタ
機　能► 現在のカレントグラフレコードへのポインタを返す。
Cの関数► Graph **GMGetGraph*(void);
返り値はグラフレコードへのポインタ。

$A133　GMCopyGraph

引　数► long　dstGraphPtr　;コピー先のグラフレコードへのポインタ
long　srcGraphPtr　;コピー元のグラフレコードへのポインタ
返り値► D0.L　リザルトコード
A0.L　コピー先のグラフレコードへのポインタが返る
機　能► srcGraphPtr で指定したグラフレコードの内容を、dstGraphPtr で指定したグラフレコードにコピーする。dstGraphPtr 用の、ビットマップレコード、ビジブルリージョン、クリップリージョン、各リージョンレコード用のブロックも作成する。すでに各リージョンレコード用のブロックが作成されていて、メンバとして登録されていた場合には、その廃棄は行われない。
疑似ポインタ可。

再配置が発生する。

Cの関数► int *GMCopyGraph*(Graph *dstGraphPtr,Graph *srcGraphPtr);
返り値はリザルトコード。

$A136 GMMoveGraph

引　数► long　pt　；グラフレクタングルを移動させるポイント（絶対位置）
返り値► D0.L　= 0
　　　　A0.L　カレントグラフレコードへのポインタ
機　能► カレントグラフレコードのグラフレクタングルの座標をグローバル座標系での pt の位置に移動させる。グラフレクタングルの大きさは変化しない。
Cの関数► void *GMMoveGraph*(LPoint pt);
返り値はない。

$A137 GMSlideGraph

引　数► long　pt　；グラフレクタングルを移動させるポイント（相対位置）
返り値► D0.L　= 0
　　　　A0.L　カレントグラフレコードへのポインタ
機　能► カレントグラフレコードのグラフレクタングルの座標をグローバル座標系での pt で示された値だけ相対移動させる。グラフレクタングルの大きさは変化しない。
Cの関数► void *GMSlideGraph*(LPoint pt);
返り値はない。

$A138 GMSetClip

引　数► long　rgnHdl　；リージョンレコードへのハンドル
返り値► D0.L　リザルトコード
　　　　A0.L　リージョンレコードへのハンドルが返る
機　能► カレントグラフレコードのクリップリージョンに、rgnHdl で指定したリージョンをセットする。
スクリプトに記録される。
疑似ハンドルも可。
再配置が発生する。
Cの関数► int *GMSetClip*(Region **rgnHdl);
返り値はリザルトコード。

$A139　GMGetClip

引　数► long　rgnHdl　；リージョンレコードへのハンドル
返り値► D0.L　リザルトコード
A0.L　リージョンレコードへのハンドルが返る
機　能► カレントグラフレコードのクリップリージョンの内容を rgnHdl で指定したリージョンレコードに返す。
再配置が発生する。
Cの関数► int *GMGetClip*(Region **rgnHdl);
返り値はリザルトコード。

$A13A　GMClipRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► カレントグラフレコードのクリップリージョンに、rectPtr で指定したレクタングルをセットする。
疑似ポインタも可。
再配置が発生する。
Cの関数► int *GMClipRect*(Rect *rectPtr);
返り値はリザルトコード。

$A13B　GMSetHome

引　数► long　homePt　；ホーム位置を示すポイント
返り値► D0.L　リザルトコード
機　能► カレントグラフレコードのホーム位置をローカル座標系の pt にする。これによって設定したホーム位置は、グラフレコード内のビットマップレクタングル、グラフレクタングル、ビジブルリージョンに反映する。
Cの関数► int *GMSetHome*(LPoint HomePt);
返り値はリザルトコード。

$A13C　GMSetGraphSize

引　数► long　sizePt　；サイズを指定するポイント
返り値► D0.L　リザルトコード
機　能► カレントグラフレコードのグラフレクタングルの大きさを、sizePt で指定した (x,y) のサイズに変更する。(x,y) のどちらかが 0 以下になった場合、カレントグラフレコードのグラフレクタングルはヌルレクタングルとなる。

Cの関数► int *GMSetGraphSize*(LPoint sizePt);
返り値はリザルトコード。

$A13D GMSetBitmap

引　数► long　bitmapPtr　; ビットマップレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► カレントグラフレコードに、bitmapPtr で指定したビットマップレコードをセットする。
疑似ポインタも可。
SX-WINDOW ver.3.0 以降からはスクリプトに記録されないようになった。
Cの関数► int *GMSetBitmap*(Bitmap *bitmapPtr);
返り値はリザルトコード。

$A13E GMLocalToGlobal

引　数► long　pt　; ローカル座標系のポイント
返り値► D0.L　グローバル座標系に変換したポイント
A0.L　カレントビットマップポインタ
機　能► pt で指定したローカル座標系のポイントを、グローバル座標系に変換して返す。
Cの関数► LPoint *GMLocalToGlobal*(LPoint pt);
返り値はグローバル座標を意味するポイント。

$A13F GMGlobalToLocal

引　数► long　pt　; グローバル座標系のポイント
返り値► D0.L　ローカル座標系に変換したポイント
A0.L　カレントビットマップポインタ
機　能► pt で指定したグローバル座標系のポイントを、ローカル座標系に変換して返す。
Cの関数► LPoint *GMGlobalToLocal*(LPoint pt);
返り値はローカル座標を意味するポイント。

$A140 GMInitPen

引　数► なし
返り値► なし
機　能► カレントグラフレコードのペンを初期化する。
スクリプトに記録される。
これによって初期化されたペンは、

- ペンモードは pset
- ペンサイズは縦 1、横 1
- ペンパターンは塗りつぶし

という状態になる。

Cの関数► void *GMInitPen*(void);
返り値はない。

$A141 GMPenShow

引　数► なし

返り値► D0.L　変更後のドローレベル

0	HIDE
1	HOW

A0.L　カレントグラフレコードへのポインタ

機　能► カレントグラフレコードのドローレベルを +1 する。

Cの関数► BOOLEAN *GMPenShow*(void);
返り値は変更後のドローレベル。

$A142 GMPenHide

引　数► なし

返り値► D0.L　変更後のドローレベル

0	HIDE
1	SHOW

A0.L　カレントグラフレコードへのポインタ

機　能► カレントグラフレコードのドローレベルを −1 する。

Cの関数► BOOLEAN *GMPenHide*(void);
返り値は変更後のドローレベル。

$A143 GMPenSize

引　数► long　sizePt　; ペンサイズを指定するポイント

返り値► D0.L　前のペンサイズを示すポイント

機　能► カレントグラフレコードのペンサイズを sizePt で指定した (x,y) サイズに変更する。
スクリプトに記録される。

Cの関数► LPoint *GMPenSize*(LPoint sizePt);

返り値は前のペンサイズを意味するポイント。

$A144　GMPenMode

引　数► word　mode　; ペンモード

上位バイト	
0	フォアグラウンドカラーで描画
1	バックグラウンドカラーで描画
2	ペンパターンで描画
3	エクステンドパターンで描画

下位バイト	
0	pset
1	and
2	or
3	xor
4	npset
5	nand
6	nor
7	nxor

(以下は、グラフィックタイプのビットマップにのみ有効)

8	add
9	add limit
10	sub
11	sub limit
12	select max
13	select min
14	blend

返り値► D0.W　前のペンモード

機　能► カレントグラフレコードのペンモードを mode に変更する。
スクリプトに記録される。

Cの関数► int *GMPenMode*(int mode);
返り値は前のペンモード。

$A145　GMPenPat

引　数► long　patPtr　; ペンパターンへのポインタ

返り値► D0.L　リザルトコード
A0.L　前のペンパターンへのポインタ

機　能► カレントグラフレコードのペンパターンを patPtr で指定したペンパターンに変更する。
スクリプトに記録される。
疑似ポインタも可。

Cの関数► unsigned short **GMPenPat*(unsigned short *patPtr);
返り値は前のペンパターンへのポインタ。

$A146　GMExPat

引　数► long　patPtr　;エクステンドパターンへのポインタ
返り値► D0.L　リザルトコード
機　能► カレントグラフレコードのエクステンドパターンを patPtr で指定したエクステンドパターンに変更する。
スクリプトに記録される。
疑似ポインタも可。
Cの関数► unsigned short **GMExPat*(unsigned short *patPtr);
返り値は前のエクステンドパターンへのポインタ。

$A147　GMForeColor

引　数► word　color　;カラーコード
返り値► D0.W　前のフォアグラウンドカラーコード
機　能► カレントグラフレコードのフォアグラウンドカラーを color に変更する。
スクリプトに記録される。
Cの関数► int *GMForeColor*(int color);
返り値は前のフォアグラウンドカラーコード。

$A148　GMBackColor

引　数► word　color　;カラーコード
返り値► D0.W　前のバックグラウンドカラーコード
機　能► カレントグラフレコードのバックグラウンドカラーを color に変更する。
スクリプトに記録される。
Cの関数► int *GMBackColor*(int color);
返り値は前のバックグラウンドカラーコード。

$A149　GMAPage

引　数► word　aPage　;テキスト VRAM へのアクセスモード
返り値► D0.W　前のアクセスモード
機　能► カレントグラフレコードのビットマップのアクセスモードを aPage に変更する。テキストタイプのグラフレコードにのみ有効。
スクリプトに記録される。

Cの関数► int *GMAPage*(int aPage);
返り値は前のアクセスモード。

$A14A GMGetLoc

引　数► なし
返り値► D0.L　現在のペン位置を示すポイント
機　能► カレントグラフレコードのペン位置をローカル座標系で返す。
Cの関数► LPoint *GMGetLoc*(void);
返り値は現在のペン位置を意味するポイント。

$A14B GMGetPen

引　数► long bufPtr　;情報が返るバッファへのポインタ
返り値► (bufPtr+$00).L　ペン位置を示すポイント
(bufPtr+$04).L　ペンサイズを示すポイント
(bufPtr+$08).W　ペンモード
(bufPtr+$0A).L　ペンパターンへのポインタ
機　能► カレントグラフレコードのペン関係の情報をまとめて返す。バッファは14バイト必要。
疑似ポインタも可。
Cの関数► void *GMGetPen*(Pen *bufPtr);
結果は Pen 型のポインタ bufPtr で示される構造体に格納される。
返り値はない。

$A14C GMSetPen

引　数► long bufPtr　;ペン関係の情報を収めたバッファへのポインタ
返り値► なし
機　能► bufPtr で指定したバッファのペン関係の情報を、カレントグラフレコードに設定する。
Cの関数► void *GMSetPen*(Pen *bufPtr);
返り値はない。

$A14D GMInitialize

引　数► なし
返り値► なし

機　能► グラフマンを初期化する。フォントのアドレスなど、内部ワークにセットされる。
Cの関数► void *GMInitialize*(void);
返り値はない。

$A14E　GMNullRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
返り値► D0.L　= 0
A0.L　レクタングルレコードへのポインタが返る
機　能► rectPtr で指定したレクタングルレコードをヌルレクタングルにする。
疑似ポインタも可。
Cの関数► void *GMNullRect*(Rect *rectPtr);
返り値はない。

$A14F　GMSizeRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
long　sizePt　；サイズを示すポイント
返り値► D0.L　結果

0	ヌルレクタングルになった
1	正常終了
-1	オーバーフロー

A0.L　レクタングルレコードへのポインタが返る
機　能► rectPtr で指定したレクタングルレコードを、sizePt で指定したサイズに変更する。
疑似ポインタも可。
Cの関数► int *GMSizeRect*(Rect *rectPtr, LPoint sizePt);
返り値は結果を意味する数値。

$A150　GMAndRects

引　数► word　rectNum　；レクタングルレコードの個数
long　resultPtr　；結果の入るレクタングルレコードへのポインタ
long　rect1Ptr　；1番目のレクタングルレコードへのポインタ
long　rect2Ptr　；2番目のレクタングルレコードへのポインタ
⋮（rectNum 個分続く）
返り値► D0.L　結果

0	ヌルレクタングル
1	正常終了
-1	エラー

機　能► rectNum で指定した個数のレクタングルの重なっている部分を resultPtr で指定したレクタングルレコードに返す。
すべてのポインタについて、疑似ポインタも可。

Cの関数► int *GMAndRects*(int rectNum, Rect *resultPtr, Rect rect1Ptr, Rect *rect2Ptr, ...);
rect1Ptr 以降の引数の数と rectNum の値は等しくなければならない。
返り値は結果を意味する数値。

$A151　　GMMoveRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
long　pt　；移動先のポイント（絶対位置）

返り値► D0.L　結果

0	ヌルレクタングル
1	正常終了
-1	オーバーフロー

A0.L　レクタングルレコードへのポインタが返る

機　能► rectPtr で指定したレクタングルを、pt で指定したポイントがホーム位置になるように移動させる。
疑似ポインタも可。

Cの関数► int *GMMoveRect*(Rect *rectPtr, LPoint pt);
返り値は結果を意味する数値。

$A152　　GMSlideRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
long　pt　；移動先のポイント（相対位置）

返り値► D0.L　結果

0	ヌルレクタングル
1	正常終了
-1	オーバーフロー

A0.L　レクタングルレコードへのポインタが返る

機　能► rectPtr で指定したレクタングルを、pt で示された値だけ相対移動させる。
疑似ポインタも可。

Cの関数► int *GMSlideRect*(Rect *rectPtr, LPoint pt);
返り値は結果を意味する数値。

$A153　GMInsetRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
long　ofsPt　；レクタングル拡大/縮小オフセットを示すポイント

返り値► D0.L　結果

0	ヌルレクタングル
1	正常終了
-1	オーバーフロー

A0.L　レクタングルレコードへのポインタが返る

機　能► rectPtr で指定したレクタングルを ofsPt で指定したオフセット分、縮小する。つまり、左上と右下の座標から、それぞれ ofsPt で示される (x,y) の値を引く。したがって、ofsPt の (x,y) として負の値を指定することで、レクタングルの拡大を行うことになる。
疑似ポインタも可。

Cの関数► int *GMInsetRect*(Rect *rectPtr, LPoint pt);
返り値は結果を意味する数値。

$A154　GMAndRect

引　数► long　resultPtr　；結果の入るレクタングルレコードへのポインタ
long　rect1Ptr　；1番目のレクタングルレコードへのポインタ
long　rect2Ptr　；2番目のレクタングルレコードへのポインタ

返り値► D0.L　結果

0	ヌルレクタングル
1	正常終了

A0.L　結果の入るレクタングルレコードへのポインタが返る

機　能► rect1Ptr と rect2Ptr で指定した2つのレクタングルの重なっている矩形領域を、resultPtr で指定したレクタングルレコードに返す。
すべてのポインタについて疑似ポインタも可。

Cの関数► int *GMAndRect*(Rect *resultPtr, Rect *rect1Ptr, Rect *rect2Ptr);
結果は Rect 型のポインタ resultPtr で示されるレクタングルレコードに格納される。
返り値は結果を意味する数値。

$A155　GMOrRect

引　数► long　resultPtr　; 結果の入るレクタングルレコードへのポインタ
long　rect1Ptr　;1番目のレクタングルレコードへのポインタ
long　rect2Ptr　;2番目のレクタングルレコードへのポインタ

返り値► D0.L　結果

0	ヌルレクタングル
1	正常終了

A0.L　結果の入るレクタングルレコードへのポインタが返る

機　能► rect1Ptr と rect2Ptr で指定した2つのレクタングルが収まる最小の矩形領域を resultPtr で指定したレクタングルレコードに返す。
すべてのポインタについて疑似ポインタも可。

Cの関数► int *GMOrRect*(Rect *resultPtr, Rect *rect1Ptr, Rect *rect2Ptr);
結果は Rect 型のポインタ resultPtr で示されるレクタングルレコードに格納される。
返り値は結果を意味する数値。

$A156　GMPtInRect

引　数► long　rectPtr　; レクタングルレコードへのポインタ
long　pt　; ポイント

返り値► D0.L　結果

0	ポイントはレクタングルの外側
1	ポイントはレクタングルの内側

機　能► pt で指定したポイントが rectPtr で指定したレクタングルの内側か外側かを調べる。
疑似ポインタも可。

Cの関数► BOOLEAN *GMPtInRect*(Rect *rectPtr, LPoint pt);
返り値は結果を意味する数値。

$A157　GMEqualRect

引　数► long　rect1Ptr　;1番目のレクタングルレコードへのポインタ
long　rect2Ptr　;2番目のレクタングルレコードへのポインタ

返り値► D0.L　結果

0	2つのレクタングルは同じでない
1	2つのレクタングルは同じ

機　能► rect1Ptr と rect2Ptr で指定した2つのレクタングルが同じ矩形領域を示しているかどうかを調べる。
すべてのポインタについて疑似ポインタも可。

Cの関数► BOOLEAN *GMEqualRect*(Rect *rect1Ptr, Rect *rect2Ptr);
返り値は結果を意味する数値。

$A158　GMEmptyRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　結果

0	レクタングルはヌルレクタングルではない
1	レクタングルはヌルレクタングル

機　能► rectPtr で指定したレクタングルがヌルレクタングルであるかどうかを調べる。
疑似ポインタも可。

Cの関数► BOOLEAN *GMEmptyRect*(Rect *rectPtr);
返り値は結果を意味する数値。

$A159　GMAdjustRect

引　数► long　resultPtr　；結果の入るレクタングルレコードへのポインタ
long　outerRectPtr　；外側になるレクタングルレコードへのポインタ
long　innerRectPtr　；内側になるレクタングルレコードへのポインタ

返り値► D0.L　結果

0	移動しなかった
1	移動した
-1	エラー

A0.L　結果の入るレクタングルレコードへのポインタが返る

機　能► innerRectPtr で指定したレクタングルを、outerRectPtr で示される矩形領域の内部に収まるように移動させ、その結果を resultPtr に返す。
すべてのポインタについて疑似ポインタも可。

Cの関数► int *GMAdjustRect*(Rect *resultPtr, Rect *outerRectPtr, Rect *innerRectPtr);
結果は Rect 型のポインタ resultPtr で示されるレクタングルレコードに格納される。
返り値は結果を意味する数値。

グラフマン

$A15A　　GMNewRgn

引　数► なし
返り値► D0.L　　リザルトコード
　　A0.L　　リージョンレコードへのハンドル
機　能► 新しいリージョンレコード用のブロックを作成し、ハンドルを返す。作成されたリージョンはヌルリージョン。
再配置が発生する。
Cの関数► Region ***GMNewRgn*(void);
返り値はリージョンレコードへのハンドル。

$A15B　　GMDisposeRgn

引　数► long　　rgnHdl　　；廃棄するリージョンレコードへのハンドル
返り値► D0.L　　リザルトコード
機　能► rgnHdl で指定したリージョンレコードを廃棄する。
Cの関数► int *GMDisposeRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A15C　　GMOpenRgn

引　数► なし
返り値► D0.L　　リザルトコード
機　能► カレントグラフレコードに対して行われる描画の、リージョンへの記録を開始する（ただし、「リージョンに記録される」と記された描画に限る）。
ドローレベルが −1 される。
再配置が発生する。
Cの関数► int *GMOpenRgn*(void);
返り値はリザルトコード。

$A15D　　GMCloseRgn

引　数► long　　rgnHdl　　；記録された結果を返すリージョンレコードへのハンドル
返り値► D0.L　　リザルトコード
　　A0.L　　リージョンレコードへのハンドルが返る
機　能► カレントグラフレコードについて行われていたリージョンの記録を終了し、記録の内容を rgnHdl で指定したリージョンレコードに返す。
ドローレベルが +1 される。

再配置が発生する。

Cの関数► int *GMCloseRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A15E　GMNullRgn

引　数► long　rgnHdl　;リージョンレコードへのハンドル
返り値► D0.L　リザルトコード
A0.L　リージョンレコードへのハンドルが返る
機　能► rgnHdl で指定したリージョンをヌルリージョンにする。
再配置が発生する。
Cの関数► int *GMNullRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A15F　GMRectRgn

引　数► long　rgnHdl　;リージョンレコードへのハンドル
long　rectPtr　;レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
A0.L　リージョンレコードへのハンドルが返る
機　能► rgnHdl で指定したリージョンを、rectPtr で指定したレクタングルの形にする。
rectPtr は疑似ポインタも可。
再配置が発生する。
Cの関数► int *GMRectRgn*(Region **rgnHdl, Rect *rectPtr);
返り値はリザルトコード。

$A160　GMCopyRgn

引　数► long　destRgnHdl　;コピー先のリージョンレコードへのハンドル
long　srcRgnHdl　;コピー元のリージョンレコードへのハンドル
返り値► D0.L　リザルトコード
A0.L　コピー先のリージョンレコードへのハンドル
機　能► srcRgnHdl で指定したリージョンの内容を destRgnHdl で指定されるリージョンレコードにコピーする。
srcRgnHdl は疑似ハンドルも可。
再配置が発生する。
Cの関数► int *GMCopyRgn*(Region **destRgnHdl, Region **srcRgnHdl);
返り値はリザルトコード。

$A161　GMMoveRgn

引　数►	long	rgnHdl	；リージョンレコードへのハンドル
	long	pt	；移動先のポイント（絶対位置）
返り値►	D0.L	リザルトコード	
	A0.L	リージョンレコードへのハンドルが返る	

機　能► rgnHdl で指定したリージョンのバウンドレクタングルを pt で指定される座標まで移動させる。
疑似ハンドルも可。

Cの関数► int *GMMoveRgn*(Region **rgnHdl, LPoint pt);
返り値はリザルトコード。

$A162　GMSlideRgn

引　数►	long	rgnHdl	；リージョンレコードへのハンドル
	long	pt	；移動先のポイント（相対位置）
返り値►	D0.L	リザルトコード	
	A0.L	リージョンレコードへのハンドルが返る	

機　能► rgnHdl で指定したリージョンのバウンドレクタングルを pt で示された値だけ相対移動させる。
疑似ハンドルも可。

Cの関数► int *GMSlideRgn*(Region **rgnHdl, LPoint pt);
返り値はリザルトコード。

$A163　GMInsetRgn

引　数►	long	rgnHdl	；リージョンレコードへのハンドル
	long	ofsPt	；リージョン拡大/縮小オフセットを示すポイント
返り値►	D0.L	結果	

0	ヌルリージョン
1	正常終了
-1	エラー

A0.L　リージョンレコードへのハンドルが返る

機　能► rgnHdl で指定したリージョンを ofsPt で指定したオフセット分、縮小する。ofsPt の (x,y) として負の値を指定することでリージョンの拡大を行うことになる。
再配置が発生する。

Cの関数► int *GMInsetRgn*(Region **rgnHdl, LPoint pt);
返り値は結果を意味する数値。

$A164　GMAndRgn

引　数► long　resultHdl　；結果の入るリージョンレコードへのハンドル
long　rgn1Hdl　；1番目のリージョンレコードへのハンドル
long　rgn2Hdl　；2番目のリージョンレコードへのハンドル

返り値► D0.L　リザルトコード
A0.L　結果の入るリージョンレコードへのハンドルが返る

機　能► rgn1Hdl と rgn2Hdl で指定した2つのリージョンの重なっている部分を、resultHdl で指定したリージョンレコードに返す。
rgn1Hdl, rgn2Hdl は疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMAndRgn*(Region **resultHdl, Region **rgn1Hdl, Region **rgn2Hdl);
結果は、Region 型のハンドル resultHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A165　GMOrRgn

引　数► long　resultHdl　；結果の入るリージョンレコードへのハンドル
long　rgn1Hdl　；1番目のリージョンレコードへのハンドル
long　rgn2Hdl　；2番目のリージョンレコードへのハンドル

返り値► D0.L　リザルトコード
A0.L　結果の入るリージョンレコードへのハンドルが返る

機　能► rgn1Hdl と rgn2Hdl で指定した2つのリージョンを合成した図形を resultHdl で指定したリージョンレコードに返す。
rgn1Hdl, rgn2Hdl は疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMOrRgn*(Region **resultHdl, Region **rgn1Hdl, Region *rgn2Hdl);
結果は、Region 型のハンドル resultHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A166　GMDiffRgn

引　数► long　resultHdl　；結果の入るリージョンレコードへのハンドル
long　rgn1Hdl　；1番目のリージョンレコードへのハンドル
long　rgn2Hdl　；2番目のリージョンレコードへのハンドル

返り値► D0.L　リザルトコード

A0.L　結果の入るリージョンレコードへのハンドルが返る

機　能► rgn1Hdl で指定したリージョンの内側で、rgn2Hdl で指定したリージョンの外側という図形を、resultHdl で指定したリージョンレコードに返す。
rgn1Hdl, rgn2Hdl は疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMDiffRgn*(Region **resultHdl,Region **rgn1Hdl,Region *rgn2Hdl);
結果は、Region 型のハンドル resultHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A167　GMXorRgn

引　数► long　resultHdl　；結果の入るリージョンレコードへのハンドル
long　rgn1Hdl　；1番目のリージョンレコードへのハンドル
long　rgn2Hdl　；2番目のリージョンレコードへのハンドル

返り値► D0.L　リザルトコード
A0.L　結果の入るリージョンレコードへのハンドルが返る

機　能► rgn1Hdl と rgn2Hdl で指定した 2 つのリージョンを XOR した図形を、resultHdl で指定したリージョンレコードに返す。
rgn1Hdl, rgn2Hdl は疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMXorRgn*(Region **resultHdl,Region **rgn1Hdl,Region *rgn2Hdl);
結果は、Region 型のハンドル resultHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A168　GMPtInRgn

引　数► long　rgnHdl　；リージョンレコードへのハンドル
long　pt　；ポイント

返り値► D0.L　結果

0	ポイントはリージョンの外側
1	ポイントはリージョンの内側
-1	エラー

機　能► pt で指定したポイントが、rgnHdl で指定したリージョンの内側か外側かを調べる。

Cの関数► int *GMPtInRgn*(Region **rgnHdl, LPoint pt);
返り値は結果を意味する数値。

$A169　　GMRectInRgn

引　数► long　rgnHdl　;リージョンレコードへのハンドル
　　　　long　rectPtr　;レクタングルレコードへのポインタ

返り値► D0.L　結果

0	リージョンとレクタングルは重ならない
1	リージョンとレクタングルは重なっている
-1	エラー

機　能► rgnHdl で指定したリージョンと rectPtr で指定したレクタングルに重なる部分があるかどうかを調べる。
疑似ハンドル、疑似ポインタも可。

Cの関数► int *GMRectInRgn*(Region **rgnHdl, Rect *rectPtr);
返り値は結果を意味する数値。

$A16A　　GMEqualRgn

引　数► long　rgn1Hdl　;1番目のリージョンレコードへのハンドル
　　　　long　rgn2Hdl　;2番目のリージョンレコードへのハンドル

返り値► D0.L　結果

0	2つのリージョンは同じではない
1	2つのリージョンは同じ
-1	エラー

機　能► rgn1Hdl と rgn2Hdl で指定した2つのリージョンが同じ図形を示しているかどうかを調べる。
すべてのリージョンについて、疑似ハンドルも可。

Cの関数► int *GMEqualRgn*(Region **rgn1Hdl, Region **rgn2Hdl);
返り値は結果を意味する数値。

$A16B　　GMEmptyRgn

引　数► long　rgnHdl　;リージョンレコードへのハンドル

返り値► D0.L　結果

0	リージョンはヌルリージョンではない
1	リージョンはヌルリージョン
-1	エラー

機　能► rgnHdl で指定したリージョンがヌルリージョンであるかどうかを調べる。
疑似ハンドル可。

Cの関数► int *GMEmptyRgn*(Region **rgnHdl);
返り値は結果を意味する数値。

$A16C　GMImgToRgn

引　数► long　rgnHdl　；リージョンレコードへのハンドル
long　bitmapPtr　；ビットマップレコードへのポインタ
long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　リージョンレコードへのハンドル

機　能► bitmapPtr で指定されたビットマップの rectPtr で指定された範囲内で、カラーコードが 0 以外の領域を求め、rgnHdl で指定したリージョンに収める。
ビットマップがテキストタイプの場合はアクセスページが参照される。
再配置が発生する。

Cの関数► int *GMImgToRgn*(Region **rgnHdl, Bitmap *bitmapPtr, Rect *rectPtr);
結果は、Region 型のハンドル rgnHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A16D　GMInitBitmap

引　数► word　scrType　；スクリーンタイプ

0	テキストタイプ
1	グラフィックタイプ

long　bitmapPtr　；ビットマップレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　ビットマップレコードへのポインタが返る

機　能► bitmapPtr で指定したビットマップを scrType で示されたスクリーンタイプで初期化する。
疑似ポインタも可。

Cの関数► int *GMInitBitmap*(int scrType, Bitmap *bitmapPtr);
返り値はリザルトコード。

$A16E　GMMove

引　数► long　pt　；移動先のポイント（絶対位置）

返り値► なし

機　能► カレントグラフレコードのペンを pt で指定したローカル座標に移動させる。

Cの関数► int *GMMove*(LPoint pt);
返り値は意味を持たない。

$A16F　　GMMoveRel

引　数► long　　pt　　　　　　　；移動先のポイント（相対位置）
返り値► なし
機　能► カレントグラフレコードのペンを pt で示した値に従って相対移動させる。
Cの関数► int *GMMoveRel*(LPoint pt);
返り値は意味を持たない。

$A170　　GMLine

引　数► long　　pt　　　　　　　；ラインの終点を示すポイント（絶対位置）
返り値► D0.L　　リザルトコード
機　能► ペンの現在位置から pt で示したポイントまで、ペンモードとペンサイズに従ってラインを引く。この結果、ペン位置は終点に移る。
スクリプトに記録される。
リージョンに記録される。
Cの関数► int *GMLine*(LPoint pt);
返り値はリザルトコード。

$A171　　GMLineRel

引　数► long　　pt　　　　　　　；ラインの終点を示すポイント（相対位置）
返り値► D0.L　　リザルトコード
機　能► ペンの現在位置から、pt で示した相対位置のポイントまで、ペンモードとペンサイズに従ってラインを引く。この結果、ペン位置は終点に移る。
スクリプトに記録される。
リージョンに記録される。
Cの関数► int *GMLineRel*(LPoint pt);
返り値はリザルトコード。

$A172　　GMFrameRect

引　数► long　　rectPtr　　　　；レクタングルレコードへのポインタ
返り値► D0.L　　リザルトコード
機　能► rectPtr で指定したレクタングルの枠を、ペンモードとペンサイズに従って描く。
スクリプトに記録される。
リージョンに記録される。
疑似ポインタも可。

Cの関数► int *GMFrameRect*(Rect *rectPtr);
返り値はリザルトコード。

$A173　GMFillRect

引　数► long　rectPtr　;レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► rectPtr で指定したレクタングルを、ペンモードに従って塗りつぶす。
スクリプトに記録される。
疑似ポインタも可。
Cの関数► int *GMFillRect*(Rect *rectPtr);
返り値はリザルトコード。

$A174　GMFrameOval

引　数► long　rectPtr　;レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► rectPtr で指定したレクタングルに内接する楕円の枠を、ペンモードとペンサイズに従って描く。
スクリプトに記録される。
リージョンに記録される。
疑似ポインタも可。
Cの関数► int *GMFrameOval*(Rect *rectPtr);
返り値はリザルトコード。

$A175　GMFillOval

引　数► long　rectPtr　;レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► RectPtr で指定したレクタングルに内接する楕円を、ペンモードに従って塗りつぶす。
スクリプトに記録される。
疑似ポインタも可。
Cの関数► int *GMFillOval*(Rect *rectPtr);
返り値はリザルトコード。

$A176　GMFrameRRect

引　数► long　rectPtr　;レクタングルレコードへのポインタ

	long	ovalPt	;楕円の x 直径、y 直径を示すポイント
返り値►	D0.L	リザルトコード	

機　能► rectPtr で指定されたレクタングルに内接し、ovalPt に示したような楕円の 4 分の 1 円周を四角に持つようなラウンドレクタングルの枠を、ペンサイズ、ペンモードに従って描く。
スクリプトに記録される。
リージョンに記録される。
疑似ポインタも可。

Cの関数► int *GMFrameRRect*(Rect *rectPtr, LPoint ovalPt);
返り値はリザルトコード。

$A177　GMFillRRect

引　数►	long	rectPtr	;レクタングルレコードへのポインタ
	long	ovalPt	;楕円の x 直径、y 直径を示すポイント
返り値►	D0.L	リザルトコード	

機　能► rectPtr で指定されたレクタングルに内接し、ovalPt に示したような楕円の 4 分の 1 円周を四角に持つようなラウンドレクタングルを、ペンモードに従って塗りつぶす。
スクリプトに記録される。
疑似ポインタも可。

Cの関数► int *GMFillRRect*(Rect *rectPtr, LPoint ovalPt);
返り値はリザルトコード。

$A178　GMFrameArc

引　数►	long	rectPtr	;円弧が内接するレクタングルレコードへのポインタ
	word	startAngle	;開始角度
	word	endAngle	;終了角度
返り値►	D0.L	リザルトコード	

機　能► rectPtr, startAngle, endAngle で指定された円弧の枠を描画する。
スクリプトに記録される。
リージョンに記録される。

Cの関数► int *GMFrameArc*(Rect *rectPtr, int startAngle, int endAngle);
返り値はリザルトコード。

$A179　GMFillArc

引　数▶ long　rectPtr　；円弧が内接するレクタングルレコードへのポインタ
　word　startAngle　；開始角度
　word　endAngle　；終了角度
返り値▶ D0.L　リザルトコード
機　能▶ rectPtr, startAngle, endAngle で指定された円弧の内部を塗りつぶす。
スクリプトに記録される。
リージョンに記録される。
Cの関数▶ int *GMFillArc*(Rect *rectPtr,int startAngle,int endAngle);
返り値はリザルトコード。

$A17A　GMFrameRgn

引　数▶ long　rgnHdl　；リージョンレコードへのハンドル
返り値▶ D0.L　リザルトコード
機　能▶ rgnHdl で指定したリージョンの枠を、ペンサイズ、ペンモードに従って描く。
スクリプトに記録される。
リージョンに記録される。
疑似ハンドルも可。
Cの関数▶ int *GMFrameRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A17B　GMFillRgn

引　数▶ long　rgnHdl　；リージョンレコードへのハンドル
返り値▶ D0.L　リザルトコード
機　能▶ rgnHdl で指定したリージョンの中をペンモードに従って塗りつぶす。
スクリプトに記録される。
疑似ハンドルも可。
Cの関数▶ int *GMFillRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A17C　GMFramePoly

引　数▶ long　polyHdl　；ポリゴンレコードへのハンドル
返り値▶ D0.L　リザルトコード
機　能▶ polyHdl で指定したポリゴンの枠を描画する。

スクリプトに記録される。
リージョンに記録される。

Cの関数► int *GMFramePoly*(Polygon **polyHdl);
返り値はリザルトコード。

$A17D GMFillPoly

引　数► long polyHdl ;ポリゴンレコードへのハンドル
返り値► D0.L リザルトコード
機　能► polyHdl で指定したポリゴンの内部を塗りつぶす。
スクリプトに記録される。
リージョンに記録される。

Cの関数► int *GMFillPoly*(Polygon **polyHdl);
返り値はリザルトコード。

$A17E GMScroll

引　数► long rectPtr ;スクロール範囲のレクタングルレコードへのポインタ
long ofsPt ;スクロールオフセットを示すポイント
long updateRgnHdl ;再描画の範囲が返るリージョンレコードへのハンドル

返り値► D0.L リザルトコード
A0.L 再描画の範囲が返るリージョンレコードへのハンドルが返る

機　能► rectPtr で指定した矩形範囲内で、ofsPt で示される値だけスクロールを行う。スクロールによって空白となった部分が updateRgnHdl で指定したリージョンレコードに返る。また、空白となった部分はバックグラウンドカラーで塗りつぶされる。
rectPtr は疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMScroll*(Rect *rectPtr, LPoint ofsPt,
Region **updateRgnHdl);
スクロールによって空白となった部分が Region 型のハンドル rgnHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A17F GMCopy

引　数► long srcBitmapPtr ;コピー元ビットマップレコードへのポインタ

long	destBitmapPtr	；コピー先ビットマップレコードへのポインタ
long	srcRectPtr	；コピー元レクタングルレコードへのポインタ
long	destRectPtr	；コピー先レクタングルレコードへのポインタ
word	copyMode	；コピーモード
long	maskRgnHdl	；マスク範囲のリージョンレコードへのハンドル

返り値► D0.L　リザルトコード

機　能► srcBitmapPtr で指定したビットマップの srcRectPtr で指定した範囲を、destBitmapPtr の destRectPtr の中の、maskRgnHdl で指定したリージョン範囲内にコピーする。コピー元とコピー先でレクタングルのサイズが異なる場合は、拡大/縮小を行う。

copyMode は、ペンモードと同様な意味を持つ。

maskRgnHdl として 0 を指定すると、マスクは行わない。

コピー先とコピー元のビットマップが同一の場合、重なり判定を行う。

コピー先のビットマップがカレントビットマップと同一の場合、グラフレクタングル、ビジブルリージョン、クリップリージョンでクリッピングを行い、スクリプトにも記録する。

すべてのポインタ、ハンドルは、疑似ポインタ、疑似ハンドルも可。

Cの関数►

```
int GMCopy(Bitmap *srcBitmapPtr, Bitmap *destBitmapPtr,
Rect *srcRectPtr, Rect *destRectPtr,
int copyMode, Region **maskRgnHdl);
```

返り値はリザルトコード。

$A180　GMCopyMask

引　数►

long	srcBitmapPtr	；コピー元ビットマップレコードへのポインタ
long	destBitmapPtr	；コピー先ビットマップレコードへのポインタ
long	maskBitmapPtr	；マスクのビットマップレコードへのポインタ
long	srcRectPtr	；コピー元レクタングルレコードへのポインタ
long	destRectPtr	；コピー先レクタングルレコードへのポインタ
long	destPt	；コピー先のレクタングルの左上の頂点を意味するポイント
long	maskRectPtr	；マスクのレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► srcBitmapPtr で指定したグラフレコードの srcRectPtr で指定した範囲を、destBitmapPtr の destPt からの矩形領域にコピーする。拡大/縮小は行わない。コピーモードは pset に固定。

maskBitmapPtr は、テキストタイプで1ページ。maskRectPtr 内のビットイメージもビットが1の部分についてコピーが行われる。

コピー先のビットマップがカレントビットマップと同一の場合、グラフレクタ

ングル、ビジブルリージョン、クリップリージョンでクリッピングを行う。
すべてのポインタは、疑似ポインタも可。

Cの関数► int *GMCopyMask*(Bitmap *srcBitmapPtr, Bitmap *destBitmapPtr, Bitmap *maskBitmapPtr, Rect *srcRectPtr, LPoint destPt, Rect *maskRectPtr);
返り値はリザルトコード。

$A182　GMPlotImg

引　数► long　plotImgPtr　；プロットイメージへのポインタ
long　rectPtr　；描画範囲のレクタングルレコードへのポインタ
word　plotMode　；プロットモード

上位8ビット	
	ページ数（0 〜 4） 0を指定すると、2を指定したことと同じになる
下位8ビット	
0	標準
1	反転
2	ハイライト
3	ハイライト反転
4	消去
6	網掛け
7	網掛け反転
8	不可視
9	不可視反転

返り値► D0.L　リザルトコード

機　能► plotImgPtr で指定したプロットイメージを、カレントグラフレコードのrect Ptr で指定した領域に描画する。描画領域は128×128ドット以内。
plotImgPtr は、疑似ポインタも可。

Cの関数► int *GMPlotImg*(unsigned short *plotImgPtr, Rect *rectPtr, int plotMode);
返り値はリザルトコード。

$A183　GMPutRImg

引　数► long　rectImgPtr　；レクタングルイメージへのポインタ
long　pt　；描画するポイント

返り値► D0.L　リザルトコード

機　能► rectImgPtr で指定したレクタングルイメージを、カレントグラフレコードの pt の座標から描画する。描画モードはpset。

テキストタイプのグラフレコードの場合、描画するページはグラフレコード内の設定に依存する。
疑似ポインタも可。

Cの関数► int *GMPutRImg*(RectImg *rectImgPtr, LPoint pt);
返り値はリザルトコード。

$A186 GMDupHImg

引　数►	long	imgPtr	; テキストタイプのビットイメージへのポインタ
	long	pt	; 描画するポイント
	word	height	; ビットイメージの y 方向のサイズ
	word	width	; 描画する x 方向のドット数

返り値► D0.L リザルトコード

機　能► imgPtr で指定したテキストタイプのビットイメージ（x 方向サイズは 1 ドット）を、カレントグラフポートの pt の座標から x 方向に width ドット繰り返して描画する。
疑似ポインタも可。

Cの関数► int *GMDupHImg*(unsigned short *imgPtr, LPoint pt, int height, int width);
返り値はリザルトコード。

$A187 GMDupVImg

引　数►	long	imgPtr	; テキストタイプのビットイメージへのポインタ
	long	pt	; 描画するポイント
	word	width	; ビットイメージの x 方向のサイズ
	word	height	; 描画する y 方向のドット数

返り値► D0.L リザルトコード

機　能► imgPtr で指定したテキストタイプのビットイメージ（y 方向サイズは 1 ドット）を、カレントグラフポートの pt の座標から y 方向に height ドット繰り返して描画する。
疑似ポインタも可。

Cの関数► int *GMDupVImg*(unsigned short *imgPtr, LPoint pt, int width, int height);
返り値はリザルトコード。

$A188　　GMDupHRImg

引　数► long　rectImgPtr　；レクタングルイメージへのポインタ
long　pt　；描画するポイント
word　width　；描画する x 方向のドット数

返り値► D0.L　リザルトコード

機　能► rectImgPtr で指定したテキストタイプのレクタングルイメージ（x 方向サイズは 16 ドット以下）を、カレントグラフレコードの pt の座標から x 方向に width ドット繰り返して描画する。
疑似ポインタも可。

Cの関数► int *GMDupHRImg*(RectImg *rectImgPtr, LPoint pt, int width);
返り値はリザルトコード。

$A189　　GMDupVRImg

引　数► long　rectImgPtr　；レクタングルイメージへのポインタ
long　pt　；描画するポイント
word　height　；描画する y 方向のドット数

返り値► D0.L　リザルトコード

機　能► rectImgPtr で指定したテキストタイプのレクタングルイメージ（y 方向サイズは 16 ドット以下）を、カレントグラフレコードの pt の座標から y 方向に height ドット繰り返して描画する。
疑似ポインタも可。

Cの関数► int *GMDupVRImg*(RectImg *rectImgPtr, LPoint pt, int height);
返り値はリザルトコード。

$A18B　　GMFontKind

引　数► word　fontKind　；フォントカインド

0	ROM12 フォント
1	ROM16 フォント
2	ROM24 フォント
3〜	フォントマンに登録されているフォント

返り値► D0.W　前のフォントカインド

機　能► カレントグラフレコードのフォントカインドを fontKind にする。
スクリプトに記録する。
再配置が発生する。

Cの関数► int *GMFontKind*(int fontKind);
返り値は前のフォントカインド。

$A18C GMFontFace

引　数► word fontFace ；フォントフェイス

bit0	ボールド
bit1	イタリック
bit2	アンダーライン
bit3	中抜き
bit4	影付き

返り値► D0.W 前のフォントフェイス
機　能► カレントグラフレコードのフォントフェイスを fontFace にする。
スクリプトに記録する。
再配置が発生する。
Cの関数► int *GMFontFace*(int fontFace);
返り値は前のフォントフェイス。

$A18D GMFontMode

引　数► word fontMode ；フォントモード

0	pset
1	and
2	or
3	xor
4	npset
5	nand
6	nor
7	nxor

+$10 で文字の背景を描画しない

返り値► D0.L 前のフォントモード
機　能► カレントグラフレコードのフォントモードを fontMode にする。
スクリプトに記録する。
再配置が発生する。
Cの関数► int *GMFontMode*(int fontMode);
返り値は前のフォントモード。

$A18E GMFontSize

引　数► long sizePt ；フォントサイズを示すポイント
返り値► D0.L 前のフォントサイズを示すポイント

機　能► カレントグラフレコードのフォントサイズを sizePt の (x,y) に設定する(128×128 ドット以内)。(0,0) を指定すると、それぞれのフォントカインドの標準のサイズになる。
スクリプトに記録する。
再配置が発生する。

Cの関数► LPoint *GMFontSize*(LPoint sizePt);
返り値は前のフォントサイズを意味するポイント。

$A18F	GMDrawChar

引　数► word　ch　; 描画するキャラクタ

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードのペン位置から ch で指定されたキャラクタを、フォントカインド、フォントフェイス、フォントモード、フォントサイズ、フォアグラウンドカラー、バックグラウンドカラーに従って描画する。
スクリプトに記録する。
再配置が発生する。

Cの関数► int *GMDrawChar*(int ch);
返り値はリザルトコード。

$A190	GMDrawStrL

引　数► long　strLPtr　; LASCII 型の文字列へのポインタ

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードのペン位置から、strLPtr で指定された LASCII 型の文字列を、フォントカインド、フォントフェイス、フォントモード、フォントサイズ、フォアグラウンドカラー、バックグラウンドカラーに従って描画する。
スクリプトに記録する。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMDrawStrL*(const _LASCII *strLPtr);
返り値はリザルトコード。

$A191	GMDrawStr

引　数► long　strPtr　; 文字列へのポインタ
long　offset　; 文字列先端からのオフセット
word　length　; 文字列のバイト数

返り値► D0.L　リザルトコード

機　能► strPtr で指定した文字列の、offset バイト目から length バイトの文字列を表示する。
描画は、カレントグラフレコードのペン位置から、フォントカインド、フォントフェイス、フォントモード、フォントサイズ、フォアグラウンドカラー、バックグラウンドカラーに従って行われる。
スクリプトに記録する。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMDrawStr*(const char *strPtr, long offset, int length);
返り値はリザルトコード。

$A192　　GMDrawStrZ

引　数► long　strZPtr　　; ASCIIZ 型の文字列へのポインタ

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードのペン位置から、strZPtr で指定された ASCIIZ 型の文字列を、フォントカインド、フォントフェイス、フォントモード、フォントサイズ、フォアグラウンドカラー、バックグラウンドカラーに従って描画する。
スクリプトに記録する。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMDrawStrZ*(const char *strZPtr);
返り値はリザルトコード。

$A194　　GMCharWidth

引　数► word　ch　　; キャラクタ

返り値► D0.L　結果/リザルトコード

上位ワード	イタリック width
下位ワード	width

機　能► ch で指定したキャラクタの横幅を、カレントグラフレコード内のフォント関係のメンバに従って求める。
再配置が発生する。

Cの関数► long *GMCharWidth*(int ch);
返り値は、上位ワードがイタリックで描画した場合の横幅、下位ワードが通常の横幅。エラーが発生した場合はリザルトコード。

$A195　GMStrLWidth

引　数► long　strLPtr　; LASCII 型の文字列へのポインタ

返り値► D0.L　結果/リザルトコード

上位ワード	イタリック width
下位ワード	width

機　能► strLPtr で指定した文字列の横幅を、カレントグラフレコード内のフォント関係のメンバに従って求める。

疑似ポインタも可。

再配置が発生する。

Cの関数► long *GMStrLWidth*(const _LASCII *strLPtr);

返り値は、上位ワードがイタリックで描画した場合の横幅、下位ワードが通常の横幅。エラーが発生した場合はリザルトコード。

$A196　GMStrWidth

引　数► long　strPtr　; 文字列へのポインタ

long　offset　; 文字列先端からのオフセット

word　length　; 文字列のバイト数

返り値► D0.L　結果/リザルトコード

上位ワード	イタリック width
下位ワード	width

機　能► strPtr で指定した文字列の、offset バイト目から length バイトの文字列の横幅を、カレントグラフレコード内のフォント関係のメンバに従って求める。

疑似ポインタも可。

再配置が発生する。

Cの関数► long *GMStrWidth*(const char *strPtr, long offset, int length);

返り値は、上位ワードがイタリックで描画した場合の横幅、下位ワードが通常の横幅。エラーが発生した場合はリザルトコード。

$A197　GMStrLength

引　数► long　strPtr　; 文字列へのポインタ

long　offset　; 文字列先端からのオフセット

word　width　; ドット数

返り値► D0.L　文字列のバイト数

機　能► カレントグラフレコード内のフォント関係のメンバに従った場合、strPtr の示す文字列の offset バイト目から何バイトを width ドットのなかで表示

できるかを求める。文字列のなかにコントロールコードがあった場合、その前の文字までのバイト数を返す。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMStrLength*(const char *strPtr, long offset, int length);
返り値は文字列のバイト数。

$A198 GMFontInfo

引　数► なし

返り値► D0.L　結果

上位ワード	width
下位ワード	height

機　能► 1文字の全角文字の幅と高さの最大値を、カレントグラフレコード内の情報に従って求める。イタリック width は返らない。
再配置が発生する。

Cの関数► LPoint *GMFontInfo*(void);
返り値は横幅と高さを意味するポイント。

$A199 GMOpenScript

引　数► long　rgnHdl　；リージョンレコードへのハンドル

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードに対して行われる描画のスクリプトへの記録を開始する（ただし、「スクリプトに記録される」と記された描画に限る）。rgnHdl で指定したリージョンがスクリプトの外形となり、このリージョンのバウンドレクタングルがスクリプトの大きさとなる。
ドローレベルが −1 される。
疑似ハンドルも可。
再配置が発生する。

Cの関数► int *GMOpenScript*(Region **rgnHdl);
返り値はリザルトコード。

$A19A GMCloseScript

引　数► long　scriptHdl　；記録された結果を返すスクリプトレコードへのハンドル

返り値► D0.L　リザルトコード

A0.L　スクリプトレコードへのハンドルが返る

機　能► カレントグラフレコードについて行われていたスクリプトの記録を終了し、記録の内容を scriptHdl で指定したスクリプトレコードに返す。 scriptHdl として 0 を指定した場合、グラフマンがブロックを確保し、ハンドルを返す。
ドローレベルが +1 される。
再配置が発生する。

Cの関数► int *GMCloseScript*(GScript **scriptHdl);
返り値はリザルトコード。

$A19B　GMDisposeScript

引　数► long　scriptHdl　;スクリプトレコードへのハンドル

返り値► D0.L　リザルトコード

機　能► scriptHdl で指定したスクリプトレコードを廃棄する。

Cの関数► int *GMDisposeScript*(GScript **scriptHdl);
返り値はリザルトコード。

$A19C　GMDrawScript

引　数► long　scriptHdl　;スクリプトレコードへのハンドル
long　rectPtr　;レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► scriptHdl で指定したスクリプトを、カレントグラフレコードに描画する。スクリプトの外形を示すリージョンのバウンドレクタングルの大きさが rectPtr で指定したレクタングルと等しくなるよう、拡大/縮小を行う。
rectPtr は疑似ポインタも可。
再配置が発生する。

Cの関数► int *GMDrawScript*(GScript **scriptHdl, Rect *rectPtr);
返り値はリザルトコード。

$A19D　GMGetScript

引　数► long　scriptHdl　;スクリプトレコードへのハンドル
long　offset　;オフセット位置

返り値► D0.L　次のコマンドへのオフセット
-1 :　エラー
A0.L　次のコマンドへのポインタ

機　能► scriptHdl で指定したスクリプトレコード内部のスクリプトデータの、先頭から offset バイト目から、次のコマンドの位置を返す。offset として 0

を指定した場合、最初のコマンドを返す。

Cの関数► long *GMGetScript*(GScript **scriptHdl, int offset);
返り値は次のコマンドへのオフセット。

$A19E　GMOpenPoly

引　数► なし
返り値► D0.L　リザルトコード
機　能► ポリゴンの記録を開始する。ドローレベルが−1され、以降、実行される $A170 GMLine で描画した座標が記録される。
再配置が発生する。
Cの関数► int *GMOpenPoly*(void);
返り値はリザルトコード。

$A19F　GMClosePoly

引　数► long　polyHdl　；ポリゴンレコードへのハンドル
返り値► D0.L　リザルトコード
A0.L　ポリゴンレコードへのハンドル
機　能► ポリゴンの記録を終了し、polyHdl で指定したポリゴンレコードに結果を返す。polyHdl として 0 を指定すると、グラフマンがヒープ上に再配置可能ブロックを作成し、そこに結果を収めてハンドルを返す。
再配置が発生する。
Cの関数► int *GMClosePoly*(Polygon **polyHdl);
返り値はリザルトコード。

$A1A0　GMDisposePoly

引　数► long　polyHdl　；ポリゴンレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► polyHdl で指定したポリゴンレコードを廃棄する。
Cの関数► int *GMDisposePoly*(Polygon **polyHdl);
返り値はリザルトコード。

$A1A1　GMShadowStrZ

引　数► long　strZPtr　；ASCIIZ 型の文字列へのポインタ
long　pt　；描画するポイント
返り値► D0.L　リザルトコード

機　能► カレントグラフレコードの pt で指定されたポイントから、strZPtr で指定した ASCIIZ 型の文字列を影付きで表示する。背景はバックグラウンドカラー、文字は黒、影は白に固定。この結果、ペン位置は最後の文字の次の位置に移動する。
疑似ポインタも可。

Cの関数► int *GMShadowStrZ*(const char *strZPtr, LPoint pt);
返り値はリザルトコード。

$A1A2　GMShadowRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードに rectPtr で指定されるレクタングルの枠を、影付きで描画する。上辺と左辺は黒、右辺と下辺は薄灰色に影が白く描画され、内部はバックグラウンドカラーで塗りつぶされる。
疑似ポインタも可。

Cの関数► int *GMShadowRect*(Rect *rectPtr);
返り値はリザルトコード。

$A1A3　GMInvertRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
word　lineStyle　；ラインスタイル

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードに rectPtr で指定されるレクタングルの枠を、lineStyle で示されたラインスタイルで、描画モード XOR で描画する。クリッピングはビットマップレクタングルとのみ行われる。
疑似ポインタも可。

Cの関数► int *GMInvertRect*(Rect *rectPtr, int lineStyle);
返り値はリザルトコード。

$A1A5　GMInvertBits

引　数► long　pt　；描画するポイント
long　bitmapPtr　；ビットマップへのポインタ

返り値► D0.L　リザルトコード

機　能► カレントグラフレコードの pt で指定した位置に、bitmapPtr で指定されたビットイメージを描画モード XOR で描画する。クリッピングはビットマップレクタングルとのみ行われる。

疑似ポインタも可。

Cの関数► int *GMInvertBits*(LPoint pt, Bitmap *bitmapPtr);

返り値はリザルトコード。

$A1A6　GMMapPt

引　数► long　pt　；オリジナルのポイント

long　srcRectPtr　；pt を含むレクタングルレコードへのポインタ

long　destRectPtr　；写像先のレクタングルレコードへのポインタ

返り値► D0.L　結果を示すポイント

機　能► srcRectPtr で示されるレクタングルとポイント pt の位置関係に相当する、destRectPtr 中のポイントを返す。srcRectPtr、destRectPtr のどちらか一方でもヌルレクタングルであった場合、結果は (0,0) となる。

すべてのポインタは、疑似ポインタも可。

Cの関数► LPoint *GMMapPt*(LPoint pt, Rect *srcRectPtr, Rect *destRectPtr);

返り値は結果を示すポイント。

$A1A7　GMMapRect

引　数► long　rectPtr　；オリジナルのレクタングル

long　srcRectPtr　；rectPtr を含むレクタングルレコードへのポインタ

long　destRectPtr　；写像先のレクタングルレコードへのポインタ

返り値► D0.L　0

A0.L　rectPtr

機　能► srcRectPtr で示されるレクタングルとレクタングル rectPtr の位置関係に相当する、destRectPtr 中のレクタングルを返す。結果は、引数として渡した rectPtr で指定されたレコードのなかに返る。srcRectPtr, destRectPtr のどちらか一方でもヌルレクタングルであった場合、結果はヌルレクタングルとなる。

すべてのポインタは、疑似ポインタも可。

Cの関数► void *GMMapRect*(Rect *rectPtr, Rect *srcRectPtr, Rect *destRectPtr);

結果は Rect 型のポインタ rectPtr で示されるレクタングルレコードに格納される。

返り値はない。

$A1A8　　GMMapPoly

引　数► long　polyHdl　　　　;ポリゴンレコードへのハンドル
　　　　long　srcRectPtr　　　;写像元のレクタングルへのポインタ
　　　　long　destRectPtr　　 ;写像先のレクタングルへのポインタ

返り値► D0.L　0
　　　　A0.L　ポリゴンレコードへのハンドル

機　能► srcRectPtr で指定されたレクタングル上にある polyHdl で指定したポリゴンを、destRectPtr 上に写像した結果を polyHdl 内に返す。どちらかのレクタングルがヌルレクタングルの場合、結果はヌルポリゴンとなる。
再配置が発生する。

Cの関数► void *GMMapPoly*(Polygon **polyHdl, Rect *srcRectPtr, Rect destRectPtr);
結果は、Polygon 型のハンドル polyHdl で示されるポリゴンレコードに格納される。
返り値はない。

$A1A9　　GMMapRgn

引　数► long　rgnHdl　　　　　;オリジナルのリージョン
　　　　long　srcRectPtr　　　; rectPtr を含むレクタングルレコードへのポインタ
　　　　long　destRectPtr　　 ;写像先のレクタングルレコードへのポインタ

返り値► D0.L　0
　　　　A0.L　rgnHdl

機　能► srcRectPtr で示したレクタングル中の rgnHdl に相当する、destRectPtr で示されるレクタングル中でのリージョンを返す。結果は、引数として渡した rgnHdl で指定されたレコードのなかに返る。srcRectPtr, destRectPtr のどちらか一方でもヌルレクタングルであった場合、結果はヌルレクタングルとなる。
すべてのポインタは疑似ポインタも可。
再配置が発生する。

Cの関数► void *GMMapRgn*(Region **rgnHdl, Rect *srcRectPtr, Rect *destRectPtr);
結果は、Region 型のハンドル rgnHdl で示されたリージョンレコードに格納される。
返り値はない。

$A1AA　GMScalePt

引　数► long　pt　;オリジナルのポイント
long　srcRectPtr　;元のレクタングルレコードへのポインタ
long　destRectPtr　;変換後のレクタングルレコードへのポインタ

返り値► D0.L　変換後のポイント

機　能► srcRectPtr で示されるレクタングルが、destRectPtr で示される大きさまで拡大/縮小された場合、srcRectPtr 中にある pt で指定されたポイントが移動する位置を返す。srcRectPtr、destRectPtr のどちらか一方でもヌルレクタングルであった場合、結果は (0,0) となる。
すべてのポインタは、疑似ポインタも可。

Cの関数► LPoint *GMScalePt*(LPoint pt, Rect *srcRectPtr, Rect *destRectPtr);
返り値は変換後のポイント。

$A1AB　GMInitPalet

引　数► なし

返り値► なし

機　能► SRAM に収められているテキストパレットデータをテキストパレットに設定する。

Cの関数► void *GMInitPalet*(void);
返り値はない。

$A1AD　GMDrawG16

引　数► long　g16ImgPtr　;G16 イメージレコードへのポインタ
long　pt　;描画するポイント

返り値► なし

機　能► グラフィックタイプのカレントグラフレコードの pt の位置から、g16ImgPtr で指定された G16 イメージを描画する。
疑似ポインタも可。

Cの関数► void *GMDrawG16*(TX16 *g16ImgPtr, LPoint pt);
返り値はない。

$A1AF　GMGetPixel

引　数► long　bitmapPtr　;ビットマップレコードへのポインタ
long　pt　;色を読み出すポイント

返り値► D0.W　　パレットコード

機　能► bitmapPtr で指定されたビットマップのなかの pt の位置のパレットコードを返す。pt がビットマップレクタングルの外なら 0 が返る。テキストタイプのビットマップの場合、ビットマップレコード内のアクセスページが参照される。

疑似ポインタも可。

Cの関数► int *GMGetPixel*(Bitmap *bitmapPtr, LPoint pt);

返り値はパレットコード。

$A1B1　　GMCalcMask

引　数► long　srcImgPtr　　; 変換元のビットイメージへのポインタ
long　destImgPtr　　; 変換後のビットイメージへのポインタ
word　srcLineBytes　　; 変換元のビットイメージの横のバイト数
word　destLineBytes　　; 変換後のビットイメージの横のバイト数
word　width　　; ビットイメージの x 方向のサイズ
word　height　　; ビットイメージの y 方向のサイズ

返り値► D0.L　　リザルトコード
A0.L　　変換後のビットイメージへのポインタが返る

機　能► srcImgPtr で指定したビットイメージから、そのマスクのビットイメージを作成し、destImgPtr で指定したビットイメージレコードに返す。srcLineBytes と destLineBytes は偶数でなければならない。srcImgPtr と destImgPtr は同じポインタを指定してはいけない。

すべてのポインタは、疑似ポインタも可。

Cの関数► int *GMCalcMask*(unsigned short *srcImgPtr, unsigned short *destImgPtr, int srcLineBytes, int destLineBytes, int width, int height);

結果は short 型のポインタ destImgPtr で示されるビットイメージレコードに格納される。

返り値はリザルトコード。

$A1B2　　GMCalcFrame

引　数► long　srcImgPtr　　; 変換元のビットイメージへのポインタ
long　destImgPtr　　; 変換後のビットイメージへのポインタ
word　srcLineBytes　　; 変換元のビットイメージの横のバイト数
word　destLineBytes　　; 変換後のビットイメージの横のバイト数
word　width　　; ビットイメージの x 方向のサイズ
word　height　　; ビットイメージの y 方向のサイズ

返り値► D0.L　　リザルトコード

A0.L　変換後のビットイメージへのポインタが返る

機　能► srcImgPtr で指定したビットイメージから、その枠のビットイメージを作成し、destImgPtr で指定したビットイメージレコードに返す。srcLineBytes と destLineBytes は偶数でなければならない。srcImgPtr と destImgPtr は同じポインタを指定してはいけない。
すべてのポインタは、疑似ポインタも可。

Cの関数► int *GMCalcFrame*(unsigned short *srcImgPtr, unsigned short *destImgPtr, int srcLineBytes, int destLineBytes, int width, int height);
結果は short 型のポインタ destImgPtr で示されるビットイメージレコードに格納される。
返り値はリザルトコード。

$A1B3　SXLongMul

引　数► long　bufPtr　；結果が返るバッファ(8バイト)へのポインタ
long　num1　；符号付ロングワード整数
long　num2　；符号付ロングワード整数

返り値► A0.L　結果が返るバッファへのポインタが返る

機　能► num1 と num2 の符号付ロングワード整数同士の積(64ビット)を、bufPtr で指定したバッファに返す。
疑似ポインタも可。

Cの関数► void **SXLongMul*(void *bufPtr, long num1, long num2);
返り値は、結果が返る2ロングワードのバッファへのポインタ。

$A1B4　SXFixRound

引　数► long　fixNum　；固定小数点数

返り値► D0.W　整数

機　能► fixNum で指定した固定小数点数の小数部を四捨五入して整数に変換する。

Cの関数► int *SXFixRound*(long fixNum);
返り値は四捨五入した結果。

$A1B6　SXFixMul

引　数► long　fixNum1　；固定小数点数1
long　fixNum2　；固定小数点数2

返り値► D0.L　積の固定小数点数
$7FFFFFFF ： オーバーフロー

$80000000 : アンダーフロー

機　能► fixNum1 と FixNum2 の 2 つの固定小数点数の積を返す。

Cの関数► long *SXFixMul*(long fixNum1, long fixNum2);
返り値は積、またはオーバー/アンダーフローを意味する数値。

$A1B7　SXFixDiv

引　数► word　num1　; 符号付ワード整数値 1
word　num2　; 符号付ワード整数値 2

返り値► D0.L　商の固定小数点数
$7FFFFFFF : ゼロで除算を行った

機　能► num1 ÷ num2 を計算し、商を固定小数点数で返す。

Cの関数► long *SXFixDiv*(int num1, int num2);
返り値は商、またはゼロで除算を行ったことを意味する数値。

$A1B8　GMGetFontTable

引　数► なし

返り値► D0.L　0
A0.L　フォントテーブルのアドレス

(A0)	12 ドット半角 ASCII
4(A0).L	12 ドット全角特殊
8(A0).L	16 ドット半角 ASCII
12(A0).L	16 ドット半角外字
16(A0).L	16 ドット 1/4 角 ASCII
20(A0).L	16 ドット 1/4 角 ASCII
24(A0).L	16 ドット 1/4 角 ASCII
28(A0).L	16 ドット全角非漢字
32(A0).L	16 ドット全角第 1 水準
36(A0).L	16 ドット全角第 2 水準
40(A0).L	16 ドット全角外字
44(A0).L	16 ドット全角特殊
48(A0).L	24 ドット半角 ASCII
52(A0).L	24 ドット半角外字
56(A0).L	24 ドット 1/4 角 ASCII
60(A0).L	24 ドット 1/4 角 ASCII
64(A0).L	24 ドット 1/4 角 ASCII
68(A0).L	24 ドット全角非漢字
72(A0).L	24 ドット全角第 1 水準
76(A0).L	24 ドット全角第 2 水準
80(A0).L	24 ドット全角外字
84(A0).L	24 ドット全角特殊

機　能► フォントの格納されているアドレスのテーブルを返す。

Cの関数► unsigned char ****GMGetFontTable*(void);

返り値はフォントテーブルへのポインタ。

$A1B9　　GMCopyStdProc

引　数► long　gProcTbl　;描画エントリテーブルを返すバッファ($40バイト)のアドレス

返り値► D0.L　0

A0.L　結果が返るバッファのアドレス

機　能► gProcTbl で指定したバッファに標準描画エントリテーブルを作成する。

Cの関数► void *GMCopyStdProc*(int (**gProcTbl)());

返り値はない。

$A1BA　　GMStrZWidth

引　数► long　strZPtr　;ASCIIZ 型の文字列へのポインタ

返り値► D0.L　結果/リザルトコード

上位ワード	イタリック width
下位ワード	width

機　能► strZPtr で指定した文字列の横幅を、カレントグラフレコード内のフォント関係のメンバに従って求める。

疑似ポインタも可。

再配置が発生する。

Cの関数► long *GMStrZWidth*(const char *strZPtr);

返り値は上位ワードがイタリックで描画した場合の横幅、下位ワードが通常の横幅。エラーが発生した場合はリザルトコード。

$A1BB　　GMTransImg

引　数► long　srcBitmapPtr　;コピー元のビットマップレコードへのポインタ

long　destBitmapPtr　;コピー先のビットマップレコードへのポインタ

long　srcRectPtr　;コピー元のレクタングルレコードへのポインタ

long　destRectPtr　;コピー先のレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► 異なるタイプのスクリーン間(srcBitmapPtr で指定されるビットマップから destBitmapPtr で指定されるビットマップ)で、ビットイメージのコピーを

行う。コピー元のビットマップ上の srcRectPtr で示されるレクタングル内部をコピー先のビットマップ上の destRectPtr で示されるレクタングル内部にコピーするが、両レクタングルのサイズが異なる場合、拡大/縮小が行われる。

コピー先のビットマップがカレントの場合、グラフレクタングル、ビジブルリージョンでクリッピングが行われる。

再配置が発生する。

Cの関数► int *GMTransImg*(Bitmap *srcBitmapPtr, Bitmap *destBitmapPtr, Rect *srcRectPtr, Rect *destRectPtr);

返り値はリザルトコード。

$A1BC　GMFillRImg

引　数► long　rectImgPtr　；レクタングルイメージレコードへのポインタ

long　pt　；描画時に左上となるポイント

返り値► D0.L　リザルトコード

機　能► rectImgPtr で指定されるテキストタイプ・1ページのレクタングルイメージを、ペンモードに従って描画する。描画位置は pt で指定する。

再配置が発生する。

Cの関数► int *GMFillRImg*(RectImg *rectImgPtr, LPoint pt);

返り値はリザルトコード。

$A1BD　GMFillImg

引　数► long　imgPtr　；イメージレコードへのポインタ

long　rectPtr　；描画位置、範囲を示すレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► imgPtr で指定されるテキストタイプ・1ページのイメージを、ペンモードに従って描画する。描画位置、範囲は rectPtr で指定する。

再配置が発生する。

Cの関数► int *GMFillImg*(unsigned short *imgPtr, Rect *rectPtr);

返り値はリザルトコード。

$A1BE　GMSlidedRgn

引　数► long　destRgnHdl　；結果が返るリージョンレコードへのハンドル

long　srcRgnHdl　；移動するリージョンレコードへのハンドル

long　pt　；移動量を意味するポイント

返り値► D0.L リザルトコード

機　能► srcRgnHdl で指定されるリージョンを pt で指定される移動量だけ相対移動し、その軌跡を destRgnHdl にリージョンとして返す。
再配置が発生する。

Cの関数► int *GMSlidedRgn*(Region **destRgnHdl, Region **srcRgnHdl, LPoint pt);
結果は、Region 型のハンドル destRgnHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A1BF　GMPaintRgn

引　数► long rgnHdl ; 結果が返るリージョンレコードへのハンドル
long bitmapPtr ; ビットマップレコードへのポインタ
long pt ; 走査を開始するポイント

返り値► D0.L リザルトコード
A0.L 結果が返るリージョンレコードへのハンドル

機　能► bitmapPtr で指定したビットマップのポイント、pt 周辺の同じ色の領域を rgnHdl で指定されたリージョンに返す。
再配置が発生する。

Cの関数► int *GMPaintRgn*(Region **rgnHdl, Bitmap *bitmapPtr, LPoint pt);
結果は Region 型のハンドル rgnHdl で示されたリージョンレコードに格納される。
返り値はリザルトコード。

$A1C0　GMSetRgnLine

引　数► word length ; リージョンデータの１行の最大値

返り値► D0.L 前の最大値

機　能► リージョンデータの１行の最大バイト数を length に設定する。length は最大 $200 までの偶数で、0 を指定すると初期状態に戻る。

Cの関数► int *GMSetRgnLine*(int length);
返り値は前の最大値。

$A1C1　GMGetRgnLine

引　数► なし

返り値► D0.L 現在設定されている１行の最大バイト数

機　能► リージョンデータの１行の最大バイト数を返す。

Cの関数► int *GMGetRgnLine*(void);

返り値は最大バイト数。

$A1C2　　GMInitGraphMode

引　数► word　　scrMode　　　　;画面モード

返り値► D0.L　　リザルトコード

機　能► scrMode で指定した画面モードにあわせてビットマップやグラフレコードの初期値を設定する。scrMode で指定するのは、IOCS $10 _CRTMOD で指定する値と同等。

Cの関数► int *GMInitGraphMode*(int scrMode);

返り値はリザルトコード。

$A1C3　　GMCurFont

引　数► なし

返り値► D0.L　　リザルトコード

A0.L　　フォントレコードへのハンドル

機　能► **システムが内部で使用するコールなので使用禁止。**

カレントグラフレコードとフォントレコードを比較して、状態が変化している場合はフォントレコードを作り直す。

再配置が発生する。

Cの関数► int *GMCurFont*(void);

返り値はリザルトコード。

$A1C4　　GMGetScrnSize

引　数► なし

返り値► D0.L　　画面の縦横のサイズを意味するポイント

機　能► 現在、設定されている画面のサイズを返す。

Cの関数► LPoint *GMGetScrnSize*(void);

返り値はサイズを意味するポイント。

$A1C5　　GMExgGraph

引　数► long　　graphPtr　　　　;グラフレコードへのポインタ

返り値► D0.L　　0

A0.L　　前のカレントグラフレコードへのポインタ

機　能► $A131 GMSetGraph と同様に、graphPtr で指定するグラフレコードをカレントにする。それまでのカレントグラフレコードへのポインタが返る。

Cの関数► Graph *GMExgGraph(Graph *graphPtr);
返り値は前のカレントグラフレコードへのポインタ。

$A1C6　　GMExgBitmap

引　数► long　　bitmapPtr　　　　; ビットマップレコードへのポインタ

返り値► D0.L　　0
A0.L　　前のビットマップレコードへのポインタ

機　能► カレントグラフレコードに bitmapPtr で指定したビットマップレコードをセットする。それまでのビットマップレコードへのポインタが返る。

Cの関数► Bitmap *GMExgBitmap(Bitmap *bitmapPtr);
返り値は前のビットマップレコードへのポインタ。

$A1C7　　GMGetBitmap

引　数► なし

返り値► D0.L　　0
A0.L　　ビットマップレコードへのポインタ

機　能► カレントグラフレコードにセットされているビットマップレコードへのポインタを返す。

Cの関数► Bitmap *GMGetBitmap(void);
返り値はビットマップレコードへのポインタ。

$A1C8　　GMCalcBitmap

引　数► long　　bitmapPtr　　　　; ビットマップレコードへのポインタ

返り値► D0.L　　リザルトコード
A0.L　　ビットマップレコードへのポインタ

機　能► bitmapPtr で指定したビットマップレコードの内容を再計算する。具体的には、bmKind, bmRect をもとにして line, page を計算する。

Cの関数► int GMCalcBitmap(Bitmap *bitmapPtr);
返り値はリザルトコード。

$A1C9　　GMCalcScrnSize

引　数► long　　bitmapPtr　　　　; ビットマップレコードへのポインタ

返り値► D0.L　　ビットマップが必要とするバイト数

機　能► bitmapPtr で指定したビットマップが必要とするメモリのバイト数を計算する。
Cの関数► long *GMCalcScrnSize*(Bitmap *bitmapPtr);
返り値はビットマップが必要とするメモリのバイト数。

$A1CA　GMNewBits

引　数► word　scrKind　；スクリーンタイプ
long　rectPtr　；ビッツの大きさを示すレクタングルレコードへのポインタ
word　aPage　；アクセスページ（テキストタイプの場合）
返り値► D0.L　リザルトコード
A0.L　ビッツへのハンドル
機　能► rectPtr で示した大きさを持つ、scrKind のタイプのビッツを作成する。
ビッツ内のビットマップレコードは base を除いて初期化される。ビッツ内のイメージデータは初期化されない。
再配置が発生する。
Cの関数► Bits ***GMNewBits*(int scrKind,Rect *rectPtr,int aPage);
返り値はビッツへのハンドル。

$A1CB　GMDisposeBits

引　数► long　bitsHdl　；ビッツへのハンドル
返り値► D0.L　0
機　能► bitsHdl で指定したビッツを廃棄する。
Cの関数► void *GMDisposeBits*(Bits **bitsHdl);
返り値はない。

$A1CC　GMLockBits

引　数► long　bitsHdl　；ビッツへのハンドル
返り値► D0.L　リザルトコード
A0.L　ビッツへのハンドル
機　能► bitsHdl で指定したビッツをロックし、ロックレベルを −1 する。
Cの関数► int *GMLockBits*(Bits **bitsHdl);
返り値はリザルトコード。

$A1CD　GMUnlockBits

引　数► long　bitsHdl　；ビッツへのハンドル

返り値► D0.L　リザルトコード
　　　　A0.L　ビッツへのハンドル
機　能► bitsHdl で指定したビッツをアンロックし、ロックレベルを +1 する。
Cの関数► int *GMUnlockBits*(Bits **bitsHdl);
　　　　返り値はリザルトコード。

$A1CE　GMItalicRect

引　数► long　rectPtr　; レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► rectPtr で指定したレクタングルをイタリックの文字を傾けるのと同じ比率だけ傾け、カレントグラフレコードに描画する。描画モードはフォントモードではなく、ペンモードが参照される。
Cの関数► int *GMItalicRect*(Rect *rectPtr);
　　　　返り値はリザルトコード。

$A1CF　GMItalicRgn

引　数► long　rgnHdl　; リージョンレコードへのハンドル
　　　　long　rectPtr　; レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
　　　　A0.L　リージョンレコードへのハンドル
機　能► rectPtr で指定したレクタングルをイタリックの文字を傾けるのと同じ比率だけ傾けた領域を、 rgnHdl で指定されたリージョンレコードに返す。
　　　　再配置が発生する。
Cの関数► int *GMItalicRgn*(Region **rgnHdl, Rect *rectPtr);
　　　　返り値はリザルトコード。

$A1D0　GMFreeBits

引　数► long　bitsHdl　; ビッツへのハンドル
返り値► D0.L　0
　　　　A0.L　ビッツへのハンドル
機　能► bitsHdl で指定したビッツのロックレベルを強制的に 0 にしてアンロックする。
Cの関数► void *GMFreeBits*(Bits **bitsHdl);
　　　　返り値はない。

$A1D1　GMCalcGraph

引　数► long　graphPtr　;グラフレコードのポインタ
返り値► D0.L　リザルトコード
A0.L　グラフレコードへのポインタ
機　能► graphPtr で指定したグラフレコードにセットされているビットマップの情報をもとにグラフレコードの内容を再計算する。ビジブルリージョン、クリップリージョン等が作成されていなかった場合、作成する。
再配置が発生する。
Cの関数► int *GMCalcGraph*(Graph *graphPtr);
返り値はリザルトコード。

$A1D2　GMPackImage

引　数► long　destPtr　;結果が納められるバッファのアドレス
long　srcPtr　;元データの先頭アドレス
long　srcLen　;元データのバイト数
返り値► D0.L　圧縮結果のバイト数/リザルトコード
A0.L　結果が納められるバッファの終端＋1
機　能► srcPtr, srcLen で与えたメモリの内容をランレングス法で圧縮し、destPtr で指定したバッファに格納する。バイト単位で同じデータが3バイト以上連続した場合に圧縮を行うので、圧縮の結果得られるデータのサイズは、最悪の場合、srcLen+((srcLen+127)/127) バイトとなる。
Cの関数► long *GMPackImage*(void *destPtr, void *srcPtr, long srcLen);
結果は、 void 型のポインタ destPtr で示されるバッファに格納される。
返り値は圧縮結果のバイト数、またはリザルトコード。

$A1D3　GMUnpackImage

引　数► long　destPtr　;結果が納められるバッファのアドレス
long　srcPtr　;圧縮データの先頭アドレス
long　srcLen　;圧縮データ展開後のバイト数
返り値► D0.L　展開したバイト数/リザルトコード
A0.L　圧縮データのバッファの終端＋1
機　能► srcPtr, srcLen で与えた圧縮データを destPtr で指定されたバッファに展開する
Cの関数► int *GMUnpackImage*(void *destPtr, void *srcPtr, long srcLen);
結果は、void 型のポインタ destPtr で示されるバッファに格納される。

返り値は、展開したバイト数またはリザルトコード。

$A1D4　GMAdjustPt

引　数► long　pt　; ポイント
long　rectPtr　; レクタングルレコードへのポインタ
返り値► D0.L　調整した結果のポイント
機　能► pt で指定したポイントが rectPtr で指定したレクタングル内に納まるように調整する。最初からレクタングルの内部にある場合は何もしない。
Cの関数► LPoint *GMAdjustPt*(LPoint pt, Rect *rectPtr);
返り値はポイント。

$A1D5　GMPutImg

引　数► long　imgPtr　; イメージレコードへのポインタ
long　rectPtr　; レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► imgPtr で指定したイメージをカレントグラフレコードの rectPtr で指定された位置と大きさで描画する。イメージのタイプはテキストタイプ、描画モードは PSET に固定。描画ページはカレントビットマップのアクセスページで決定される。
Cの関数► int *GMPutImg*(unsigned short *imgPtr, Rect *rectPtr);
返り値はリザルトコード。

$A1D6　GMCenterRect

引　数► long　destRectPtr　; 結果が返るレクタングルレコードへのポインタ
long　srcRectPtr　; 元になるレクタングルレコードへのポインタ
long　sizePt　; 新しく作るレクタングルの大きさを意味するポイント
word　mode　; レクタングルがはみ出した場合の処理

0	元になるレクタングルと新しく作るレクタングルの中心をあわせる
1	新しく作るレクタングルの左上の座標が必ず元のレクタングルの内側に納まるようにする

返り値► D0.L　結果

0	ヌルレクタングルになった
1	正常終了
-1	エラー

A0.L　結果が返るレクタングルレコードへのポインタ

機　能► srcRectPtr で指定するレクタングルの中央に sizePt で指定する大きさのレクタングルを作成し、destRectPtr に格納する。srcRectPtr がヌルレクタングルの場合、結果もヌルレクタングルとなる。

Cの関数► int *GMCenterRect*(Rect *destRectPtr, Rect *srcRectPtr, LPoint sizePt, int mode);
返り値は結果を意味する数値。

$A1D7　GMScrewRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► rectPtr で指定したレクタングルを外形とする擬似ダイアログを、カレントグラフレコードに描画する。
レクタングルとして (xs, ys)−(xe, ye) を指定した場合、ビスの位置は
(xs+4, ys+4)+(xs+14, ys+14)、(xe−15, ys+4)−(xe−5, ys+14)、
(xs+4, ye−15)−(xs+14, ye−5)、(xe−15, ye−15)−(xe−5, ye−5)
の 4 カ所となる。

Cの関数► int *GMScrewRect*(Rect *rectPtr);
返り値はリザルトコード。

$A1D8　GMAndRectRgn

引　数► long　destRgnHdl　；結果が返るリージョンレコードへのハンドル
long　srcRgnHdl　；リージョンレコードへのハンドル
long　srcRectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　結果が返るリージョンレコードへのハンドル

機　能► srcRectPtr で指定したレクタングルと srcRgnHdl で指定したリージョンの AND をとり、結果を destRgnHdl にリージョンとして格納する。共通部分がない場合、結果はヌルリージョンとなる。
再配置が発生する。

Cの関数► int *GMAndRectRgn*(Region **destRgnHdl,Region *srcRgnHdl, Rect *srcRectPtr);
結果は、Region 型のハンドル destRgnHdl で示されるリージョンレコードに格納される。

返り値はリザルトコード。

$A1D9　GMOrRectRgn

引　数► long　destRgnHdl　; 結果が返るリージョンレコードへのハンドル
long　srcRgnHdl　; リージョンレコードへのハンドル
long　srcRectPtr　; レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　結果が返るリージョンレコードへのハンドル

機　能► srcRectPtr で指定したレクタングルと srcRgnHdl で指定したリージョンの OR をとり、結果を destRgnHdl にリージョンとして格納する。
再配置が発生する。

Cの関数► int *GMOrRectRgn*(Region **destRgnHdl,Region **srcRgnHdl, Rect *srcRectPtr);
結果は、Region 型のハンドル destRgnHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A1DA　GMDiffRectRgn

引　数► long　destRgnHdl　; 結果が返るリージョンレコードへのハンドル
long　srcRgnHdl　; リージョンレコードへのハンドル
long　srcRectPtr　; レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　結果が返るリージョンレコードへのハンドル

機　能► srcRgnHdl で指定したリージョンの内側で、srcRectPtr で指定したレクタングルの外側の部分を求め、結果を destRgnHdl にリージョンとして格納する。
再配置が発生する。

Cの関数► int *GMDiffRectRgn*(Region **destRgnHdl,Region **srcRgnHdl, Rect *srcRectPtr);
結果は、Region 型のハンドル destRgnHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A1DB　GMXorRectRgn

引　数► long　destRgnHdl　; 結果が返るリージョンレコードへのハンドル
long　srcRgnHdl　; リージョンレコードへのハンドル

long　srcRectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

A0.L　結果が返るリージョンレコードへのハンドル

機　能► srcRectPtr で指定したレクタングルと srcRgnHdl で指定したリージョンの XOR をとり、結果を destRgnHdl にリージョンとして格納する。
再配置が発生する。

Cの関数► int *GMXorRectRgn*(Region **destRgnHdl,Region **srcRgnHdl, Rect *srcRectPtr);
結果は、Region 型のハンドル destRgnHdl で示されるリージョンレコードに格納される。
返り値はリザルトコード。

$A1DC　GMCharKind

引　数► word　ch　；文字コード（ASCII コード/シフト JIS コード）

返り値► D0.L　文字種コード

機　能► ch で指定した文字の種類を返す。存在しない文字コードを指定した場合、ミッシングキャラクタとなり、全角特殊として分類される。
文字種コードの分類は以下のとおり。

文字種コード	文字の種類	コードの範囲
0	半角 ASCII	$00 ～ $FF, $8000 ～ $80FF
1	半角外字	$F400 ～ $F5FF
2	1/4 角上付	$F000 ～ $F1FF
3	1/4 角下付	$F200 ～ $F3FF
4	未定義	
5	全角非漢字	$8140 ～ $84BE
6	全角第 1 水準漢字	$889F ～ $989E
7	全角第 2 水準漢字	$989F ～ $EB9E
8	全角外字	$EB9F ～ $EC9E
9	全角特殊	$84BF ～ $889E, $F600 ～ $FFFF

Cの関数► int *GMCharKind*(int ch);
返り値は文字種コード。

$A1DD　GMDiffRgnRect

引　数► long　destRgnHdl　；結果が返るリージョンレコードへのハンドル

long　srcRectPtr　；レクタングルレコードへのポインタ

long　srcRgnHdl　；リージョンレコードへのハンドル

返り値► D0.L　リザルトコード

A0.L　結果が返るリージョンレコードへのハンドル

機　能► srcRectPtr で指定したレクタングルの内側で、srcRgnHdl で指定したリー

ジョンの外側の部分を求め、結果を destRgnHdl にリージョンとして格納する。

再配置が発生する。

Cの関数► int *GMDiffRgnRect*(Region **destRgnHdl, Rect *srcRectPtr, Region **srcRgnHdl);

結果は、Region 型のハンドル destRgnHdl で示されるリージョンレコードに格納される。

返り値はリザルトコード。

$A1E0　GMAddFont

引　数► long　fLink　；フォントリンクのアドレス

返り値► D0.L　リザルトコード

機　能► **フォントマンの追加により使用禁止。**

fLink で指定したフォントリンクをシステムに追加する。

$A1E1　GMRemoveFont

引　数► long　fKind　；フォントカインド

返り値► D0.L　リザルトコード

機　能► **フォントマンの追加により使用禁止。**

fKind で指定したフォントカインドとして登録されているフォントを取り外す。

$A1E2　GMGetFontLink

引　数► なし

返り値► D0.L　フォントリンクの先頭が納められているアドレス

A0.L　フォントリンクの先頭が納められているアドレス

機　能► **フォントマンの追加により使用禁止。**

フォントリンクの先頭が格納されているアドレスを返す。

$A1E3　GMGetHProcTbl

引　数► なし

返り値► D0.L　水平描画の初期化ルーチンのテーブルのアドレス

A0.L　水平描画の初期化ルーチンのテーブルのアドレス

機　能► 水平描画の初期化ルーチンのテーブルを返す。

Cの関数► int (***GMGetHProcTbl*(void))();

返り値は、水平描画の初期化ルーチンのテーブルへのポインタ。

$A1E6　GMGetStdProcTbl

引　数► なし
返り値► D0.L　標準描画ルーチンのテーブルのアドレス
　A0.L　標準描画ルーチンのテーブルのアドレス
機　能► 標準描画ルーチンのテーブルを返す。
Cの関数► int (***GMGetStdProcTbl*(void))();
　返り値は、標準描画ルーチンのテーブルへのポインタ。

$A1E7　GMGetFontProcTbl

引　数► なし
返り値► D0.L　文字描画ルーチンのテーブルのアドレス
　A0.L　文字描画ルーチンのテーブルのアドレス
機　能► 文字描画ルーチンのテーブルを返す。
Cの関数► int (***GMGetFontProcTbl*(void))();
　返り値は、文字描画ルーチンのテーブルへのポインタ。

$A1E8　GMGetRgnProcTbl

引　数► なし
返り値► D0.L　リージョン1行演算ルーチンのテーブルのアドレス
　A0.L　リージョン1行演算ルーチンのテーブルのアドレス
機　能► リージョン1行演算ルーチンのテーブルを返す。
Cの関数► void (***GMGetRgnProcTbl*(void))();
　返り値は、リージョン1行演算ルーチンのテーブルへのポインタ。

$A1E9　GMDrawGsOne　2.0

引　数► long　gsEnvPtr　;スクリプト環境レコードへのポインタ
返り値► D0.L　リザルトコード
機　能► gsEnvPtr で指定したスクリプト環境レコードの内容に従ってスクリプトを1コマンド実行する。$A19C GMDrawScript の下請けコール。
　スクリプトのコマンド（64番以降）を追加する場合、このコールのベクタをフックする。
　再配置が発生する。
Cの関数► int *GMDrawGsOne*(GSOneEnv *gsEnvPtr);
　返り値はリザルトコード。

グラフマン

$A1EA GMPtInImg 2.0

引　数► long rectImgPtr ；レクタングルイメージレコードへのポインタ
long pt ；調べる座標を意味するポイント
word page ；ページ番号
返り値► D0.L 結果

0	座標はイメージのなかにはない
1	座標はイメージのなかにある
-1	エラー

機　能► pt で指定した座標が、rectImgPtr で指定したテキストタイプのレクタングルイメージのなかにあるかどうかを調べる。このとき、page で指定したページで、指定した座標の上下左右のドットが ON になっていれば、イメージ内部と判断される。
Cの関数► int *GMPtInImg*(RectImg *rectImgPtr, LPoint pt, int page);
返り値は結果を意味する数値。

$A1EB GMFrameNPoly 2.0

引　数► long npListPtr ；ポリゴンリストへのポインタ
long npEnvPtr ；ポリゴン環境レコードへのポインタ
返り値► D0.L リザルトコード
機　能► ニューポリゴンの枠を描画する。
再配置が発生する。
Cの関数► int *GMFrameNPoly*(NPoly *npListPtr, NPolyEnv *npEnvPtr);
返り値はリザルトコード。

$A1EC GMFillNPoly 2.0

引　数► long npListPtr ；ポリゴンリストへのポインタ
long npEnvPtr ；ポリゴン環境レコードへのポインタ
返り値► D0.L リザルトコード
機　能► ニューポリゴンの内部を塗りつぶす。
再配置が発生する。
Cの関数► int *GMFillNPoly*(NPoly *npListPtr, NPolyEnv *npEnvPtr);
返り値はリザルトコード。

$A1ED GMNPolyFrRgn 2.0

引　数► long rgnHdl ；結果が返るリージョンレコードへのハンドル

long　npListPtr　；ポリゴンリストへのポインタ
long　npEnvPtr　；ポリゴン環境レコードへのポインタ
long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　結果が返るリージョンレコードへのハンドル

機　能► npListPtr と npEnvPtr で表現されるニューポリゴンの枠を意味するリージョンを求めて rgnHdl で示されるリージョンレコードに格納する。このとき、rectPtr で示されるレクタングルでクリッピングを行う。
rgnHdl として 0 を指定した場合、グラフマンがメモリを確保する。
rectPtr として 0 を指定した場合、クリッピングを行わない。
再配置が発生する。

Cの関数► Region ***GMNPolyFrRgn*(Region **rgnHdl,NPoly *npListPtr, NPolyEnv *npEnvPtr, Rect *rectPtr);
返り値はリージョンレコードへのハンドル。
エラーが発生した場合、NULL が返る。

$A1EE　GMNPolyFlRgn　2.0

引　数► long　rgnHdl　；結果が返るリージョンレコードへのハンドル
long　npListPtr　；ポリゴンリストへのポインタ
long　npEnvPtr　；ポリゴン環境レコードへのポインタ
long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　結果が返るリージョンレコードへのハンドル

機　能► npListPtr と npEnvPtr で表現されるニューポリゴンの内部を意味するリージョンを求めて rgnHdl で示されるリージョンレコードに格納する。このとき、rectPtr で示されるレクタングルでクリッピングを行う。
閉じていないニューポリゴンは、始点と終点を直線で結んで閉じる。
rgnHdl として 0 を指定した場合、グラフマンがメモリを確保する。
rectPtr として 0 を指定した場合、クリッピングを行わない。
再配置が発生する。

Cの関数► Region ***GMNPolyFlRgn*(Region **rgnHdl,NPoly *npListPtr, NPolyEnv *npEnvPtr, Rect *rectPtr);
返り値はリージョンレコードへのハンドル。
エラーが発生した場合、NULL が返る。

$A1EF　GMPtInNPoly　2.0

引　数► long　npListPtr　；ポリゴンリストへのポインタ
long　npEnvPtr　；ポリゴン環境レコードへのポインタ

	long	pt	;調べる座標を意味するポイント
返り値►	D0.L	結果	

0	座標はニューポリゴンのなかにはない
1	座標はニューポリゴンのなかにある

機　能► pt で指定した座標が npListPtr, npEnvPtr で指定したニューポリゴンの内部にあるかどうかを調べる。
このコールは高速化のため厳密なチェックを行わないので、厳密さが要求される場合は $A1EE GMNPolyFlRgn と $A168 GMPtInRgn を使用する必要がある。

Cの関数► BOOLEAN *GMPtInNPoly*(NPoly *npListPtr,NPolyEnv *npEnvPtr, LPoint pt);
返り値は結果を意味する数値。

$A1F0　GMPtOnNPoly　2.0

引　数►	long	npListPtr	;ポリゴンリストへのポインタ
	long	npEnvPtr	;ポリゴン環境レコードへのポインタ
	long	pt	;調べる座標を意味するポイント
返り値►	D0.L	結果	

0	座標はニューポリゴンの線上にはない
1	座標はニューポリゴンの線上にある

機　能► pt で指定した座標が npListPtr, npEnvPtr で指定したニューポリゴンの線上にあるかどうかを調べる。
このコールは高速化のため厳密なチェックを行わないので、厳密さが要求される場合は $A1ED GMNPolyFrRgn と $A168 GMPtInRgn を使用する必要がある。

Cの関数► BOOLEAN *GMPtOnNPoly*(NPoly *npListPtr,NPolyEnv *npEnvPtr, LPoint pt);
返り値は結果を意味する数値。

$A1F1　GMRecordScript　2.0

引　数►	long	dataHdl	;スクリプトデータへのハンドル
	long	offset	;スクリプト先頭からのオフセット
	long	length	;記録するバイト数
返り値►	D0.L	リザルトコード	

機　能► dataHdl で指定したメモリブロック中の先頭から offset バイト目から length バイトを、カレントグラフレコードで記録中のスクリプトに記録する。
再配置が発生する。

Cの関数► int *GMRecordScript*(Handle dataHdl, long offset, long

length);
返り値はリザルトコード。

$A1F2　GMNLineRel　2.0

引　数► long　pt　；終点を示すポイント（相対位置）
long　npEnvPtr　；ポリゴン環境レコードへのポインタ
返り値► D0.L　リザルトコード
機　能► ペンの現在位置から pt で示した相対位置のポイントまで、npEnvPtr で示されるポリゴン環境レコードの内容に従ってラインを引く。
再配置が発生する。
Cの関数► int *GMNLineRel*(LPoint pt, NPolyEnv *npEnvPtr);
返り値はリザルトコード。

$A1F3　GMNLine　2.0

引　数► long　pt　；終点を示すポイント（絶対位置）
long　npEnvPtr　；ポリゴン環境レコードへのポインタ
返り値► D0.L　リザルトコード
機　能► ペンの現在位置から pt で示したポイントまで、npEnvPtr で示されるポリゴン環境レコードの内容に従ってラインを引く。
再配置が発生する。
Cの関数► int *GMNLine*(LPoint pt, NPolyEnv *npEnvPtr);
返り値はリザルトコード。

$A1F4　GMRecordPoly　2.0

引　数► long　pt　；記録する座標を意味するポイント
返り値► D0.L　リザルトコード
機　能► pt で示された座標をポリゴンに記録する。
最初の点の場合は、現在のペンの座標とあわせて 2 点分、記録する。
Cの関数► int *GMRecordPoly*(LPoint pt);
返り値はリザルトコード。

$A455　GMDitherImg　3.1

引　数► long　imgPtr　；イメージレコードへのポインタ
long　destRectPtr　；描画位置、範囲を示すレクタングルレコードへのポインタ

```
long    hv             ; イメージのサイズを指定するポイント
word    mode           ; 描画モード
word    page           ; ページ数
long    dPattern       ; ディザパターンレコードへのポインタ
```

返り値► D0.L　リザルトコード

機　能► imgPtr で指定される page ページのテキストタイプのイメージを、destRectPtr で指定される位置にモノクロディザ変換を行いながら描画する。mode で指定した描画モードが使用される。

元になるイメージのサイズはポイント hv で指定する。拡大/縮小が発生した場合、拡大率によってディザパターンレコード中の適切なディザパターンが使用される。

ディザパターンレコードの内容は以下のとおり。

```
typedef struct {
/* 最大倍率（2 なら 2 倍までの拡大時に使用される）*/
short  mag ;
/* ページ数（3 なら 8 色分のイメージが続く）*/
short  page;
/*     unsigned short img[]; 白、ライトグレー、ダークグレー、黒 ...
             に対応する 16×16 のテキストタイプ 1 ページのイメージが並ぶ */
}DPatPac;

typedef struct {
/* 総バイト数 (4+(4+32×2^n)×m) */
/* n はページ数。 m はパターン数 (m ≤ 16) */
             long  recordSize;
/*           DPatPac pattern2;  1 組のディザパターンパック
                                   用意したい倍率の数だけ並ぶ */
}DPattern;
```

dPattern として 0 を指定した場合、デフォルトのディザパターンが使用される。デフォルトのディザパターンは、白 ～ 黄の 8 色（3 ページ）に対応するパターンが 1~2 倍と 2~5 倍の 2 セットで構成されている。

スクリプトに記録される。

再配置が発生する。

Cの関数► int *GMDitherImg*(unsigned short *imgPtr,Rect *destRectPtr, LPoint hv, short mode, short page, DPattern *dPattern);

返り値はリザルトコード。

$A540　GMSetFlattness　2.0

引　数► word　theFlattness　; 近似誤差判定値

返り値► D0.L　前の近似誤差判定値

機　能► ベジェ、Bスプラインの処理で参照される似誤差判定値をセットする。
似誤差判定値の単位は、グラフマンの座標系のポイント。
デフォルトは1。

Cの関数► int *GMSetFlattness*(int theFlattness);
返り値は前の近似誤差判定値。

$A541　GMGetFlattness　2.0

引　数► なし

返り値► D0.L　近似誤差判定値

機　能► 現在の近似誤差判定値を返す。

Cの関数► int *GMGetFlattness*(void);
返り値は近似誤差判定値。

$A542　GMSetBSDepth　2.0

引　数► word　theDepth　；最大再帰階数

返り値► D0.L　前の最大再帰階数

機　能► **プリントマン用につき使用禁止。**
ベジェ、Bスプラインの処理で参照される最大再帰階数をセットする。

$A543　GMGetBSDepth　2.0

引　数► なし

返り値► D0.L　現在の最大再帰階数

機　能► **プリントマン用につき使用禁止。**
現在の最大再帰階数を返す。

$A544　GMDrawBezier　2.0

引　数► long　c1Pt　；方向点1を意味するポイント
long　c2Pt　；方向点2を意味するポイント
long　a2Pt　；終点を意味するポイント

返り値► D0.L　両端点を含む、補間されたポイントの数

機　能► ペンの現在位置を起点として、c1Pt、c2Pt を経て a2Pt に至る3次のベジェ曲線を1セグメント描画する。
スクリプトに記録される。
ポリゴンに記録される。

Cの関数► int *GMDrawBezier*(LPoint c1Pt, LPoint c2Pt, LPoint a2Pt);
返り値は補間されたポイントの数。

$A545　GMDrawBSpline　2.0

引　数► long　cPt　; 中間点を意味するポイント
long　a2Pt　; 終点を意味するポイント
返り値► D0.L　両端点を含む、補間されたポイントの数
機　能► ペンの現在位置を起点として、cPt を経て a2Pt に至る2次のBスプライン曲線を1セグメント描画する。
スクリプトに記録される。
ポリゴンに記録される。
Cの関数► int *GMDrawBSpline*(LPoint cPt, LPoint a2Pt);
返り値は補間されたポイントの数。

$A546　GMSplitBezier　2.0

引　数► long　c1Pt　; 方向点1を意味するポイント
long　c2Pt　; 方向点2を意味するポイント
long　a2Pt　; 終点を意味するポイント
long　aDivision　; 分割比を意味する固定小数
long　aSplited　; 結果を格納するバッファへのポインタ
返り値► D0.L　リザルトコード
機　能► ペンの現在位置を起点として、c1Pt, c2Pt を経てa2Ptに至る3次のベジェ曲線1セグメントを分割比 aDivision(整数部16ビット ＋ 小数部16ビット)で分割し、その2セグメントのポイント列を aSplited で示されるバッファに格納する。
Cの関数► int *GMSplitBezier*(LPoint c1Pt, LPoint c2Pt, LPoint a2Pt, long aDivision, SpltBz *aSplited);
返り値はリザルトコード。

$A547　GMSplitBSpline　2.0

引　数► long　cPt　; 中間点を意味するポイント
long　a2Pt　; 終点を意味するポイント
long　aDivision　; 分割比を意味する固定小数
long　aSplited　; 結果を格納するバッファへのポインタ
返り値► D0.L　リザルトコード
機　能► ペンの現在位置を起点として、cPt を経て a2Pt に至る2次のBスプライン

曲線1セグメントを分割比 aDivision（整数部16ビット ＋ 小数部16ビット）で分割し、その2セグメントのポイント列を aSplited で示されるバッファに格納する。

Cの関数► int *GMSplitBSpline*(LPoint cPt,LPoint a2Pt,long aDivision, SpltBSp *aSplited);
返り値はリザルトコード。

$A548　GMPtOnBezier　2.0

引　数►

long	c1Pt	；方向点1を意味するポイント
long	c2Pt	；方向点2を意味するポイント
long	a2Pt	；終点を意味するポイント
long	pt	；調べる点を意味するポイント

返り値► D0.L　指定した点による曲線の分割比を意味する固定小数
負の数の場合はエラー。

機　能► pt で指定したポイントが、現在のペン位置、c1Pt，c2Pt，a2Pt で指定した3次のベジェ曲線1セグメント上にあるかどうかを調べる。曲線上にある場合は、そのポイントによってセグメントが分割される比率を固定小数（整数部16ビット ＋ 小数部16ビット）で返す。

Cの関数► long *GMPtOnBezier*(LPoint c1Pt, LPoint c2Pt, LPoint a2Pt, LPoint pt);
返り値は、分割比を意味する固定小数または負の数。

$A549　GMPtOnBSpline　2.0

引　数►

long	cPt	；中間点を意味するポイント
long	a2Pt	；終点を意味するポイント
long	pt	；調べる点を意味するポイント

返り値► D0.L　指定した点による曲線の分割比を意味する固定小数
負の数の場合はエラー。

機　能► pt で指定したポイントが、現在のペン位置、cPt，a2Pt で指定した2次のBスプライン曲線1セグメント上にあるかどうかを調べる。曲線上にある場合は、そのポイントによってセグメントが分割される比率を固定小数（整数部16ビット ＋ 小数部16ビット）で返す。

Cの関数► long *GMPtOnBSpline*(LPoint cPt, LPoint a2Pt, LPoint pt);
返り値は、分割比を意味する固定小数または負の数。

$A54A　GMSetBSError　2.0

引　数► long　errConst　；判定範囲誤差を意味する固定小数
返り値► D0.L　前の errConst
機　能► **ユーザ使用禁止コール。**
$A548 GMPtOnBezier，$A549 GMPtOnBSpline で参照される反転範囲誤差を errConst にする。
デフォルトは $c000。

$A54B　GMGetBSError　2.0

引　数► なし
返り値► D0.L　現在の errConst
機　能► **ユーザ使用禁止コール。**
$A548 GMPtOnBezier，$A549 GMPtOnBSpline で参照される現在の反転範囲誤差を返す。

$A550　GMSetGSDraw　2.0

引　数► word　gsNo　；スクリプトコマンド番号 (0 ～ 63)
long　procPtr　；描画ルーチンへのポインタ
返り値► D0.L　リザルトコード
A0.L　前の描画ルーチンへのポインタ
機　能► gsNo で指定したスクリプトコマンドの描画ルーチンを登録する。
Cの関数► int (*GMSetGSDraw(int gsNo, int (*procPtr)()))(GSOneEnv *gsEnvPtr);
返り値は、前の描画ルーチンへのポインタ。
負の数の場合はエラー。

$A551　GMGetGSDraw　2.0

引　数► word　gsNo　；スクリプトコマンド番号 (0～63)
返り値► D0.L　リザルトコード
A0.L　現在の描画ルーチンへのポインタ
機　能► gsNo で指定したスクリプトコマンドの描画ルーチンへのポインタを返す。
Cの関数► int (*GMGetGSDraw(int gsNo))(GSOneEnv *gsEnvPtr);
返り値は描画ルーチンへのポインタ。
負の数の場合はエラー。

$A552 GMSetGSGet 2.0

引　数► word　gsNo　; スクリプトコマンド番号 (0 ～ 63)
　　　　long　procPtr　; オフセット処理ルーチンへのポインタ
返り値► D0.L　リザルトコード
　　　　A0.L　前のオフセット処理ルーチンへのポインタ
機　能► gsNo で指定したスクリプトコマンドのオフセット処理ルーチンを登録する。
Cの関数► int *GMSetGSGet*(int gsNo, int (*procPtr)());
　　　　返り値は処理ルーチンへのポインタ。
　　　　負の数の場合はエラー

$A553 GMGetGSGet 2.0

引　数► word　gsNo　; スクリプトコマンド番号 (0 ～ 63)
返り値► D0.L　リザルトコード
　　　　A0.L　現在のオフセット処理ルーチンへのポインタ
機　能► gsNo で指定したスクリプトコマンドのオフセット処理ルーチンを返す。
Cの関数► int (**GMGetGSGet*(int gsNo))();
　　　　返り値は処理ルーチンへのポインタ。
　　　　負の数の場合はエラー。

$A554 GMTileRImg 3.0

引　数► long　rectImgPtr　; レクタングルイメージレコードへのポインタ
　　　　long　rectPtr　; レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► rectPtr で指定したレクタングルの範囲内に、rectImgPtr で指定したレクタングルイメージを敷きつめる。イメージのスクリーンカインドは、カレントグラフレコードにセットされているビットマップと同じでなければならない。スクリプトに記録される。
Cの関数► int *GMTileRImg*(RectImg *rectImgPtr, Rect *rectPtr);
　　　　返り値はリザルトコード。

$A555 GMTileImg 3.0

引　数► long　imgPtr　; イメージレコードへのポインタ
　　　　long　rectPtr　; レクタングルレコードへのポインタ
　　　　long　sizePt　; ビットイメージのサイズを意味するポイント
返り値► D0.L　リザルトコード

機　能► rectPtr で指定したレクタングルの範囲内に、imgPtr で指定したイメージを敷きつめる。イメージの縦横のサイズは sizePt で指定する。イメージのスクリーンカインドは、カレントグラフレコードにセットされているビットマップと同じでなければならない。
スクリプトに記録される。

Cの関数► int *GMTileImg*(unsigned short *imgPtr, Rect *rectPtr, LPoint sizePt);
返り値はリザルトコード。

$A556　GMSetDispOffset　3.0

引　数► long　ofsPt　；縦横のオフセットを意味するポイント

返り値► なし

機　能► **グラフィックウィンドウ専用コールにつき、使用禁止。**
グラフィック画面のハードウェアスクロールのオフセット値として ofsPt をセットする。

Cの関数► void *GMSetDispOffset*(LPoint ofsPt);
返り値はない。

$A557　GMGetDispOffset　3.0

引　数► なし

返り値► D0.L　オフセットテーブルのアドレス
A0.L　オフセットテーブルのアドレス

(A0).L	ページ #0 のオフセットを意味するポイント
4(A0).L	ページ #1 のオフセットを意味するポイント
8(A0).L	ページ #2 のオフセットを意味するポイント
12(A0).L	ページ #3 のオフセットを意味するポイント

機　能► グラフィック画面のハードウェアスクロールのオフセットテーブルを返す。

Cの関数► LPoint **GMGetDispOffset*(void);
返り値はオフセットテーブルへのポインタ。

$A558　GMTestScrKindG　3.0

引　数► word　scrKind　；スクリーンカインド

返り値► D0.L　結果

0	使用不可
1	使用可能

機　能► 現在のグラフィック画面のモードで scrKind で指定したグラフィック系ビットマップが使用可能かどうかを調べる。scrKind として指定できるのは G_GRP(1)

または G_GRX(4) のみ。

Cの関数► int *GMTestScrKindG*(int scrKind);

返り値は、使用可能かどうかを意味する数値。

$A559 GMGetScrKindG 3.0

引　数► なし

返り値► D0.L　現在のグラフィック画面の種類

機　能► 現在のグラフィック画面の種類を返す。今のところ、G_GRP(1) または G_GRX(4) が返る。

Cの関数► int *GMGetScrKindG*(void);

返り値は現在のグラフィック画面の種類。

$A55D GMGetGraphMode 3.0

引　数► なし

返り値► D0.L　画面モード

機　能► $A1C2 GMInitGraphMode で初期化時に指定された画面モードを返す。

Cの関数► int *GMGetGraphMode*(void);

返り値は画面モード。

$A55E GMSetPalet 3.0

引　数► long　palHdl　;パレットハンドル

返り値► D0.L　前のカレントパレットハンドル

A0.L　前のカレントパレットハンドル

機　能► カレントパレットハンドルを palHdl に設定する。

Cの関数► Palet **GMSetPalet*(Palet **palHdl);

返り値はパレットハンドル。

$A55F GMGetPalet 3.0

引　数► なし

返り値► D0.L　現在のカレントパレットハンドル

A0.L　現在のカレントパレットハンドル

機　能► 現在のカレントパレットハンドルを返す。

Cの関数► Palet ***GMGetPalet*(void);

返り値はパレットハンドル。

$A560　GMCopy2　3.0

引　数► long　rscID　; 色変換モジュールのリソース ID
long　srcBitmapPtr　; コピー元ビットマップレコードへのポインタ
long　destBitmapPtr　; コピー先ビットマップレコードへのポインタ
long　srcRectPtr　; コピー元レクタングルレコードへのポインタ
long　destRectPtr　; コピー先レクタングルレコードへのポインタ
long　srcPalHdl　; コピー元のパレットハンドル
long　destPalHdl　; コピー先のパレットハンドル
long　palamLen　; パラメータのバイト数
long　palamPtr　; パラメータへのポインタ

返り値► D0.L　リザルトコード

機　能► rscID で指定した色変換モジュールを用いて色変換を行いながらビットマップ間で矩形領域のコピーを行う。−1 を指定すると、デフォルトの色変換モジュールが使用される。

srcBitmapPtr で指定したビットマップの、srcRectPtr で指定した範囲を、destBitmapPtr の destRectPtr にコピーする。コピー元とコピー先でレクタングルのサイズが異なる場合は、拡大/縮小を行う。

色変換モジュールが付加情報を要求する場合、palamLen，palamPtr で指定する。

カレントグラフレコードのビットマップへのコピーの場合、スクリプトに記録される。

Cの関数► int *GMCopy2*(int rscID, Bitmap *srcBitmapPtr,
Bitmap *destBitmapPtr,Rect *srcRectPtr,Rect *destRectPtr,
Palet **srcPalHdl, Palet **destPalHdl, int palamLen,
int *palamPtr);

返り値はリザルトコード。

$A562　GMMakeGrpBitmap　3.0

引　数► なし

返り値► D0.L　= 0　エラー
≠ 0　テキストタイプのビットマップレコードへのポインタ
A0.L　D0 と同様

機　能► 画面モードに応じたグラフィックタイプのビットマップレコードを作成し、カレントグラフレコードにセットする。前にセットされていたビットマップレコードへのポインタが返される。

カレントグラフレコードにセットされているビットマップはテキストタイプでなければならない。

セットされるグラフィックタイプのビットマップは、必要がなくなったら $A02F MMPtrDispose で廃棄する必要がある。

Cの関数► Bitmap **GMMakeGrpBitmap*(void);
返り値はビットマップレコードへのポインタ。

$A563 GMDrawScript2 3.0

引　数► long scriptHdl ;スクリプトへのハンドル
long rectPtr ;レクタングルレコードへのポインタ
long paletHdl ;パレットハンドル

返り値► D0.L リザルトコード

機　能► scriptHdl で示されるカラー情報の含まれたスクリプト (ver.2) を描画する。
従来のスクリプト (ver.1) も描画可能。
描画中は、paletHdl がカレントパレットにセットされる。

Cの関数► int *GMDrawScript2*(GScript **scriptHdl, Rect *rectPtr, Palet **paletHdl);
返り値はリザルトコード。

$A564 GMRecordVer 3.0

引　数► word scriptVer ;スクリプトのバージョン番号

返り値► D0.L リザルトコード

機　能► **アプリケーションは使用禁止。**
現在記録中のスクリプトのバージョン番号を scriptVer にする。

Cの関数► int *GMRecordVer*(int scriptVer);
返り値はリザルトコード。

$A565 GMForeRGB 3.0

引　数► long rgbPtr ;RGB レコードへのポインタ

返り値► D0.L カレントグラフレコードにセットした値
負の数の場合はエラー

機　能► カレントパレットを参照して、rgbPtr で指定した色に近い色をフォアグラウンドカラーとしてセットする。
RGB レコードの形式は以下のとおり。

```
typedef struct CRGB {
      unsigned short   cRed;    /* 赤 0-65535 */
      unsigned short   cGreen;  /* 緑 0-65535 */
      unsigned short   cBlue;   /* 青 0-65535 */
      unsigned short   cAlpha;  /* 拡張用 */
} CRGB;
```

スクリプトに記録される。

Cの関数► int *GMForeRGB*(CRGB *rgbPtr);
返り値は、カレントグラフレコードにセットした値。

$A566　GMBackRGB　3.0

引　数► long　rgbPtr　;RGBレコードへのポインタ
返り値► D0.L　カレントグラフレコードにセットした値
負の数の場合はエラー
機　能► カレントパレットを参照して、rgbPtr で指定した色に近い色をバックグラウンドカラーとしてセットする。
スクリプトに記録される。
Cの関数► int *GMBackRGB*(CRGB *rgbPtr);
返り値は、カレントグラフレコードにセットした値。

$A567　GMRecEnv　3.0

引　数► long　bitmapPtr　;ビットマップレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► 記録中のスクリプトに bitmapPtr で指定したビットマップレコードの情報を記録する。bitmapPtr として0を指定すると、カレントグラフレコードのビットマップを記録する。
$A199 GMOpenScript 時には、自動的にこのコールが呼び出される。
Cの関数► int *GMRecEnv*(Bitmap *bitmapPtr);
返り値はリザルトコード。

$A568　GMRecPalet　3.0

引　数► long　paletHdl　;パレットハンドル
返り値► D0.L　リザルトコード
機　能► 記録中のスクリプトに paletHdl で指定したパレットの情報を記録する。
paletHdl として0を指定すると、カレントパレットを記録する。
Cの関数► int *GMRecPalet*(Palet **paletHdl);
返り値はリザルトコード。

$A569　GMFillRImg2　3.0

引　数► long　rectImgPtr　；レクタングルイメージレコードへのポインタ
long　destRectPtr　；レクタングルレコードへのポインタ
word　mode　；描画モード
返り値► D0.L　リザルトコード
機　能► rectImgPtr で示されるレクタングルイメージ（テキストタイプ、1ページ）を、フォアグラウンドカラー、バックグラウンドカラーに従って mode で指定した描画モードで描画する。mode はフォントモードに準拠する。
元の大きさと destRectPtr で示されるレクタングルの大きさが異なる場合、拡大/縮小する。
スクリプトに記録される。
Cの関数► int *GMFillRImg2*(RectImg *rectImgPtr, Rect *destRectPtr, int mode);
返り値はリザルトコード。

$A56A　GMFillImg2　3.0

引　数► long　imgPtr　；イメージレコードへのポインタ
long　destRectPtr　；レクタングルレコードへのポインタ
word　width　；イメージの x 方向のドット数
word　height　；イメージの y 方向のドット数
word　mode　；描画モード
返り値► D0.L　リザルトコード
機　能► imgPtr で示されたビットイメージ（テキストタイプ、1ページ）をフォアグラウンドカラー、バックグラウンドカラーに従って mode で指定した描画モードで描画する。mode はフォントモードに準拠する。
width, height で示される元の大きさと、destRectPtr で示されるレクタングルの大きさが異なる場合、拡大/縮小を行う。
スクリプトに記録される。
Cの関数► int *GMFillImg2*(unsigned short *imgPtr, Rect *destRectPtr, int width, int height, int mode);
返り値はリザルトコード。

$A56B　GMSetPutID　3.0

引　数► word　rscID　；色変換モジュールのリソース ID
返り値► D0.L　前の色変換モジュールの ID
機　能► デフォルトの色変換モジュールを rscID で指定する。rscID として −1 を指定すると、現在の色変換モジュールのリソース ID を返す。

デフォルトの色変換モジュールは、スクリプトコマンド GS_PUT3 を実行するときに使用される。
デフォルトでは 0 が設定されている。

Cの関数► int *GMSetPutID*(int rscID);
返り値は色変換モジュールの ID。

$A56C GMMakePalet 3.0

引　数► word scrType ; ビットマップのスクリーンタイプ
word aPage ; アクセスページ

返り値► D0.L リザルトコード
A0.L パレットハンドル
エラーの場合は NULL

機　能► scrType で指定されたスクリーンタイプに従ってパレットハンドルを作成して返す。スクリーンタイプがテキストタイプの場合のみ aPage が意味を持つ。

Cの関数► Palet *******GMMakePalet*(int scrType, int aPage);
返り値はパレットハンドル。
エラーが発生した場合、NULL が返る。

$A56D GMFontRealSize 3.0

引　数► なし

返り値► D0.L フォントのサイズを意味するポイント

機　能► カレントグラフレコードに設定されているフォントカインド、フォントサイズをもとに装飾のないフォントの実際のサイズを返す。

Cの関数► LPoint *GMFontRealSize*(void);
返り値はポイント。

$A56E GMGetCPDFInfo 3.0

引　数► word rscID ; 色変換モジュールのリソース ID

返り値► D0.L リザルトコード
A0.L 色変換モジュール情報レコードへのポインタ

(A0).L	リソース ID
4(A0)	モジュール名（ASCIIZ 型、32 バイト）
36(A0)	コメント（ASCIIZ 型、220 バイト）

機　能► rscID で指定した色変換モジュールに関する情報を返す。色変換モジュール情報レコードが必要なくなった場合、 $A02F MMPtrDispose で廃棄する必要がある。

Cの関数► CpdfInfo **GMGetCPDFInfo*(int rscID);

返り値は色変換モジュール情報レコードへのポインタ。
エラーが発生した場合、NULL が返る。

$A56F GMGetCPDFList 3.0

引　数► なし

返り値► D0.L 色変換モジュールの数/リザルトコード
A0.L 色変換モジュール情報レコードのリストへのポインタ

機　能► すべての色変換モジュールについての情報を返す。色変換モジュール情報レコードのリストは必要がなくなった場合、$A02F MMPtrDispose で廃棄する必要がある。
リストの終端には-1L が付加される。

Cの関数► int *GMGetCPDFList*(CpdfInfo **list);
色変換モジュール情報レコードのリストへのポインタが cpdfList 型のポインタ list に格納される。
返り値は、色変換モジュールの数またはリザルトコード。

$A570 GMScanScript 3.0

引　数►	long	scriptHdl	; スクリプトへのハンドル
	long	offset	; 調べ始めるオフセット
	word	gsNo	; スクリプトコマンド番号

返り値► D0.L スクリプト先頭からのオフセット
= -1 エラー
A0.L スクリプトへのポインタ

機　能► scriptHdl で指定したスクリプトの offset バイト目から gsNo で指定したスクリプトコマンドを探し、そのオフセットを返す。発見できなかった場合は -1 が返る。

Cの関数► int *GMScanScript*(GScript **scriptHdl,int offset,int gsNo);
返り値は、スクリプト先頭からのオフセットまたはリザルトコード。

$A571 GMGetGSInfo 3.0

引　数► long scriptHdl ; スクリプトへのハンドル

返り値► D0.L = 0 エラー
≠ 0 スクリプト情報レコードへのポインタ
A0.L スクリプト情報レコードへのポインタ

(A0).W	対象とするスクリーンタイプ
2(A0)	バウンドレクタングル
10(A0).W	アクセスページ
12(A0).W	スクリプトのバージョン
14(A0).L	全体のバイト数

機　能► scriptHdl で指定したスクリプトに関する情報を返す。スクリプト情報レコードは必要がなくなった場合、$A02F MMPtrDispose で廃棄する必要がある。

Cの関数► GsInfo **GMGetGSInfo*(GScript **scriptHdl);
返り値はスクリプト情報レコードへのポインタ。
エラーが発生した場合、NULL が返る。

$A572　GMMovePoly　3.0

引　数► long　polyHdl　；ポリゴンレコードへのハンドル
long　pt　；移動先のポイント（絶対位置）

返り値► D0.L　リザルトコード

機　能► polyHdl で指定したポリゴンを、pt で指定したポイントがホーム位置になるように移動させる。

Cの関数► int *GMMovePoly*(Polygon **polyHdl, LPoint pt);
返り値はリザルトコード。

$A573　GMSlidePoly　3.0

引　数► long　polyHdl　；ポリゴンレコードへのハンドル
long　pt　；移動先のポイント（相対位置）

返り値► D0.L　リザルトコード

機　能► polyHdl で指定したポリゴンを、pt で示された値だけ相対移動させる。

Cの関数► int *GMSlidePoly*(Polygon **polyHdl, LPoint pt);
返り値はリザルトコード。

$A574　GMNewBits2　3.0

引　数► word　scrKind　；スクリーンタイプ
long　rectPtr　；ビッツの大きさを示すレクタングルレコードへのポインタ
word　aPage　；アクセスページ（テキストタイプの場合）
long　baseAddr　；ベースアドレス

返り値► D0.L　リザルトコード
A0.L　ビッツへのハンドル

機　能► rectPtr で示した大きさを持つ、scrKind のタイプのビッツを作成する。

baseAddr として 0 以外を指定すると、ビッツのイメージ領域は確保されず、ロックされた際にはそのアドレスからイメージ領域として使用する。0 を指定した場合は$A1CA GMNewBits と同じ動作をする。
再配置が発生する。

Cの関数► Bits ***GMNewBits2*(int scrKind,Rect *rectPtr,int aPage, int baseAddr);
返り値はビッツへのハンドル。
エラーが発生した場合、NULL が返る。

ライブラリ　GMSetRect4

返り値► = 0　エラー
≠ 0　正常終了

機　能► rectptr で示されるポインタのアドレスに、左上の点を（left, top）で、右下の点を（right, bottom）で与えられた値を持つ Rect 型を作成する。

Cの関数► int *GMSetRect4*(Rect *rectptr,int left,int top,int right, int bottom);

ライブラリ　GMSetRect2

返り値► = 0　エラー
≠ 0　正常終了

機　能► rectptr で示されるポインタのアドレスに、左上の点を pt1 で、右下の点を pt2 で与えられた値を持つ Rect 型を作成する。

Cの関数► int *GMSetRect2*(Rect *rectptr, LPoint pt1, LPoint pt2);

ライブラリ　GMPtToRect

返り値► = 0　エラー
≠ 0　正常終了

機　能► rectptr で示されるポインタのアドレスに、点 pt1, pt2 で示された座標から作られるレクタングル Rect 型を作成する。
pt1, pt2 の位置関係は自由。

Cの関数► int*GMPtToRect*(Rect *rectptr, LPoint pt1, LPoint size);

ライブラリ　GMGetRectHV

返り値► = 0　エラー
≠ 0　縦、横のサイズ

機　能► rectptr で与えられたレクタングルの縦、横のサイズを計算して返す。

Cの関数► long *GMGetRectHV*(Rect *rectptr);

ライブラリ　GMGetRectH

返り値► = 0　エラー

≠ 0　横のサイズ

機　能► rectptr で与えられたレクタングルの横のサイズを計算して返す。

Cの関数► long *GMGetRectH*(Rect *rectptr);

ライブラリ　GMGetRectV

返り値► = 0　エラー

≠ 0　縦のサイズ

機　能► rectptr で与えられたレクタングルの縦のサイズを計算して返す。

Cの関数► long *GMGetRectV*(Rect *rectptr);

ウィンドウマン

#include <WINDOW.H>

$A1F8　WMInit

引　数► なし

返り値► D0.L　リザルトコード

機　能► ウィンドウマンを初期化する。メモリマン、イベントマンが初期化され、リソースファイル 'SYSTEM.LB' がオープンされている必要がある。
再配置が発生する。

Cの関数► int *WMInit*(void);
返り値はリザルトコード。

$A1F9　WMOpen

引　数►

long	winPtr	; ウィンドウレコードへのポインタ
long	boundRectPtr	; レクタングルレコードへのポインタ
long	titleLStrPtr	;LASCII 型の文字列へのポインタ
word	visible	; 可視フラグ
word	wDefID	; リソース 'WDEF' の ID × 16 + オプション

	オプション
bit0	スクロールバー
bit1	クリップ
bit2	ドライブ表示
bit3	サイズボタン

long	behindWinPtr	; ウィンドウレコードへのポインタ
word	cBttn	; クローズボタンフラグ
long	taskID	; タスク ID 番号

返り値► D0.L　リザルトコード
A0.L　ウィンドウレコードへのポインタ

機　能► 新しいウィンドウを開く。
winPtr は、新しく作成するウィンドウのウィンドウレコードが納められるアドレスを示す。0 を指定すると、ウィンドウマンがヒープ上に作成する。
titleLStrPtr は、ウィンドウタイトルを示す LASCII 型の文字列へのポインタ。
visible として-1 を指定するとウィンドウは可視となり、0 を指定すると不可視となる。

behindWinPtr は、このウィンドウの上に表示されるウィンドウのレコードへのポインタを指定する。-1 を指定すると最も手前に、0 を指定すると最も奥に置かれる。
cBttn として-1 を指定するとクローズボタンを描画する。0 の場合は描画しない。
再配置が発生する。

Cの関数► Window **WMOpen*(Window *winPtr, Rect *boundRectPtr, const _LASCII *titleLStrPtr, BOOLEAN visible, int wDefID, Window *behindWinPtr, BOOLEAN cBttn, long taskID);
返り値はウィンドウレコードへのポインタ。
エラーの場合は NULL が返る。

$A1FA　　WMRefer

引　数► word　windID　; リソース 'WIND' の ID
long　winPtr　; ウィンドウレコードへのポインタ
long　behindWinPtr　; ウィンドウレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　ウィンドウレコードへのポインタ

機　能► windID で指定したウィンドウテンプレートの内容に従って新しいウィンドウを開く。
winPtr は、新しく作成するウィンドウのウィンドウレコードへのポインタを示す。0 を指定すると、ウィンドウマンがヒープ上に作成する。
behindWinPtr は、このウィンドウの上に表示されるウィンドウのレコードへのポインタを指定する。-1 を指定すると最も手前に、0 を指定すると最も奥に置かれる。
再配置が発生する。

Cの関数► Window **WMRefer*(int windID, Window *winPtr, Window *behindWinPtr);
返り値はウィンドウレコードへのポインタ。
エラーの場合は NULL が返る。

$A1FB　　WMClose

引　数► long　winPtr　; ウィンドウレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► winPtr で指定したウィンドウを閉じ、ウィンドウリストから削除する。
ウィンドウレコードをヒープ上以外に作成していた場合に使用する。
再配置が発生する。

Cの関数► int *WMClose*(Window *winPtr);
返り値はリザルトコード。

$A1FC　WMDispose

引　数► long　winPtr　; ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウを閉じ、ウィンドウリストから削除したのち、ウィンドウレコードとして確保されていたブロックを破棄する。
ウィンドウレコードをヒープ上に作成していた場合に使用する。
再配置が発生する。
Cの関数► int *WMDispose*(Window *winPtr);
返り値はリザルトコード。

$A1FD　WMFind

引　数► long　pt　; ポイント（グローバル座標）
返り値► D0.L　リザルトコード/パートコード

0	ウィンドウの外
1	システム予約
2	システム予約
3	ウィンドウコンテンツ
4	ドラッグリージョン
5	システム予約
6	サイズボタン
7	クローズボタン
8	ズームボタン（ズームイン）
9	ズームボタン（ズームアウト）
10	システム予約
11	システム予約
12	システム予約
13	矢印
14	クリップ (ON)
15	クリップ (OFF)
16	ドライブ/コントロールボタン（時計）

A0.L　ウィンドウレコードへのポインタ
機　能► pt で指定したポイントが、どのウィンドウの、どの部分にあたるかを調べ、該当するウィンドウレコードへのポインタとパートコードを返す。
Cの関数► int *WMFind*(LPoint pt, Window **winPtr);
返り値はパートコード。
Window 型のポインタ変数 winPtr に、該当するウィンドウのウィンドウレコードへのポインタが返る。

$A1FE　　WMSelect

引　数► long　winPtr　　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウをアクティブにする。
再配置が発生する。
Cの関数► int *WMSelect*(Window *winPtr);
返り値はリザルトコード。

$A1FF　　WMSelect2

引　数► long　winPtr　　;ウィンドウレコードへのポインタ
long　behindWinPtr　　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウを、behindWinPtr で指定したウィンドウの後ろに配置する。behindWinPtr として-1 を指定すると最も手前に、0 を指定すると最も奥に置かれる。
$A1FE WMSelect と異なり、サブウィンドウを消去しない。
再配置が発生する。
Cの関数► int *WMSelect2*(Window *winPtr, Window *behindWinPtr);
返り値はリザルトコード。

$A200　　WMCarry

引　数► long　winPtr　　;ウィンドウレコードへのポインタ
long　behindWinPtr　　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウを、behindWinPtr で指定したウィンドウの後ろに配置する。behindWinPtr として-1 を指定すると最も手前に、0 を指定すると最も奥に置かれる。
再配置が発生する。
Cの関数► int *WMCarry*(Window *winPtr, Window *behindWinPtr);
返り値はリザルトコード。

$A201　　WMShine

引　数► long　winPtr　　;ウィンドウレコードへのポインタ
word　value　　;設定値

0	ウィンドウをインアクティブにする
-1	ウィンドウをアクティブにする
other	パートコード

返り値► D0.L　リザルトコード

機　能► winPtr で指定したウィンドウのアクティブ/インアクティブの設定を行う。複数のウィンドウをアクティブにしてはならない。

value としてパートコードを指定した場合は、対応するパートが反転表示される。反転を元に戻す場合は、再度、このコールで反転させるか、ウィンドウ自体をインアクティブにする。

再配置が発生する。

Cの関数► int *WMShine*(Window *winPtr, int value);

返り値はリザルトコード。

$A202　WMMove

引　数►

long	winPtr	; ウィンドウレコードへのポインタ
long	pt	; ウィンドウの位置を示すポイント（グローバル座標）
word	active	; アクティブフラグ

返り値► D0.L　リザルトコード

機　能► winPtr で指定したウィンドウを pt で指定する位置に移動する。その際、active が-1 ならウィンドウをアクティブにしてから移動させる。

指定したウィンドウが子ウィンドウを持つ場合、子ウィンドウも同時に移動する。

再配置が発生する。

Cの関数► int *WMMove*(Window *winPtr, LPoint pt, BOOLEAN active);

返り値はリザルトコード。

$A203　WMSize

引　数►

long	winPtr	; ウィンドウレコードへのポインタ
long	sizePt	; サイズを示すポイント
word	fUpdate	; アップデートフラグ

返り値► D0.L　リザルトコード

機　能► winPtr で指定したウィンドウのサイズを、sizePt で示す x, y 方向のサイズに変更する。fUpdate に-1 を指定するとアップデートイベントが発生する。0 の場合、アップデートイベントは発生しない。

再配置が発生する。

Cの関数► int *WMSize*(Window *winPtr,LPoint sizePt,BOOLEAN fUpdate);

返り値はリザルトコード。

$A204　WMGrow

引　数► long　winPtr　；ウィンドウレコードへのポインタ
long　pt　；アニメーション開始ポイント（グローバル座標）
long　rectPtr　；サイズ制限を示すレクタングルレコードへのポインタ

返り値► D0.L　サイズを意味するポイント/リザルトコード

機　能► マウスの左ボタンが押されている間、マウスの動きにサイズをあわせて winPtr で指定したウィンドウの輪郭を描画する。左ボタンが離された時点で、決定したサイズを返す。結果的にサイズに変更がなかった場合は 0 が返る。

通常、pt としてはマウスが押された座標、つまり、サイズボタン内のポイントを指定する。

rectPtr で示したレクタングルレコードは、ウィンドウのサイズの下限/上限を意味する。疑似ポインタも可。左上を示すポイントがウィンドウの最小の縦横サイズを意味し、右下を示すポイントが最大の縦横サイズを意味する。

再配置が発生する。

Cの関数► LPoint *WMGrow*(Window *winPtr,LPoint pt,Rect *rectPtr);
返り値は、サイズを意味するポイントまたはリザルトコード。

$A205　WMDrag

引　数► long　winPtr　；ウィンドウレコードへのポインタ
long　pt　；アニメーション開始ポイント（グローバル座標）
long　rectPtr　；移動制限を示すレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► マウスの左ボタンが押されている間、マウスの動きにあわせて winPtr で指定したウィンドウの輪郭を移動させる。左ボタンが離された時点で、実際にウィンドウを移動させて終了する。

通常、pt としてはマウスが押された座標、つまり、ドラッグリージョン内のポイントを指定する。

rectPtr で示したレクタングルレコードは、ウィンドウの移動可能な領域を意味する（グローバル座標）。疑似ポインタも可。

再配置が発生する。

Cの関数► int *WMDrag*(Window *winPtr, LPoint pt, Rect *rectPtr);
返り値はリザルトコード。

$A206　WMZoom

引　数►	long	winPtr	;ウィンドウレコードへのポインタ
	word	partCode	;パートコード
	word	active	;アクティブフラグ
返り値►	D0.L	リザルトコード	

機　能► winPtrで指定したウィンドウを、partCodeに従ってズームイン/アウトする。partCodeはズームボックス関係(8,7)のみ有効。
activeとして-1を指定すると、ウィンドウをアクティブにしてからズームイン/アウトを行う。
再配置が発生する。

Cの関数► int *WMZoom*(Window *winPtr,int partCode,BOOLEAN active);
返り値はリザルトコード。

$A207　WMShow

引　数►	long	winPtr	;ウィンドウレコードへのポインタ
返り値►	D0.L	リザルトコード	

機　能► winPtrで指定したウィンドウを可視にする。
再配置が発生する。

Cの関数► int *WMShow*(Window *winPtr);
返り値はリザルトコード。

$A208　WMHide

引　数►	long	winPtr	;ウィンドウレコードへのポインタ
返り値►	D0.L	リザルトコード	

機　能► winPtrで指定したウィンドウを不可視にする。
再配置が発生する。

Cの関数► int *WMHide*(Window *winPtr);
返り値はリザルトコード。

$A209　WMShowHide

引　数►	long	winPtr	;ウィンドウレコードへのポインタ
	word	vFlag	;可視フラグ
返り値►	D0.L	リザルトコード	

機　能► winPtrで指定したウィンドウを、vFlagが 0 のとき不可視に、-1のとき可視にする。

再配置が発生する。

Cの関数► int *WMShowHide*(Window *winPtr, BOOLEAN vFlag);
返り値はリザルトコード。

$A20A　WMCheckBox

引　数► long　winPtr　；ウィンドウレコードへのポインタ
long　pt　；マウスのクリック座標（グローバル座標）
word　partCode　；パートコード

返り値► D0.L　結果

0	ウィンドウアイテム外でマウスが離された
-1	ウィンドウアイテム内でマウスが離された
other	リザルトコード

機　能► winPtrで指定されたウィンドウ内のpartCodeで指定されたウィンドウアイテムについて、マウスの左ボタンが離されるまで待ち、離されたポイントが指定したウィンドウアイテム内であるかどうかを調べる。マウスカーソルがウィンドウアイテム上にある間は反転表示する。
通常、ptとしてはマウスの左ボタンが押された座標、つまり、ウィンドウアイテム内のポイントを指定する。
再配置が発生する。

Cの関数► int *WMCheckBox*(Window *winPtr,LPoint pt,int partCode);
返り値は結果を意味する数値。

$A20B　WMCheckCBox

引　数► long　winPtr　；ウィンドウレコードへのポインタ
long　pt　；マウスのクリック座標（グローバル座標）

返り値► D0.L　結果

0	クローズボタン外でマウスが離された
-1	クローズボタン内でマウスが離された
other	リザルトコード

機　能► winPtrで指定されたウィンドウのクローズボックスについて、マウスの左ボタンが離されるまで待ち、離されたポイントが指定したクローズボタン内であるかどうかを調べる。マウスカーソルがクローズボタン上にある間は反転表示する。
通常、ptとしてはマウスの左ボタンが押された座標、つまり、クローズボタン内のポイントを指定する。
再配置が発生する。

Cの関数► int *WMCheckCBox*(Window *winPtr, LPoint pt);
返り値は結果を意味する数値。

$A20C　WMDrawGBox

引　数► long　winPtr　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウのサイズボタンを描画する。
再配置が発生する。
Cの関数► int *WMDrawGBox*(Window *winPtr);
返り値はリザルトコード。

$A20D　WMUpdate

引　数► long　winPtr　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウのアップデート処理を開始する。
再配置が発生する。
Cの関数► int *WMUpdate*(Window *winPtr);
返り値はリザルトコード。

$A20E　WMUpdtOver

引　数► long　winPtr　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウのアップデート処理を終了する。
再配置が発生する。
Cの関数► int *WMUpdtOver*(Window *winPtr);
返り値はリザルトコード。

$A20F　WMActive

引　数► なし
返り値► A0.L　アクティブウィンドウのウィンドウレコードへのポインタ
機　能► アクティブウィンドウのウィンドウレコードへのポインタを返す。
Cの関数► Window **WMActive*(void);
返り値はウィンドウレコードへのポインタ。

$A218　WMAddRect

引　数► long　rectPtr　;レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード

機　能► カレントウィンドウのアップデートリージョンに、rectPtr で指定したレクタングルを加える。このコールを呼ぶのは$A20D WMUpdate の前でなければならない。
レクタングルはグローバル座標で指定すること。
再配置が発生する。

Cの関数► int *WMAddRect*(Rect *rectPtr);
返り値はリザルトコード。

$A219　　WMAddRgn

引　数► long　　rgnHdl　　　　　; リージョンレコードへのハンドル

返り値► D0.L　　リザルトコード

機　能► カレントウィンドウのアップデートリージョンに、rgnHdl で指定したリージョンを加える。このコールを呼ぶのは $A20D WMUpdate の前でなければならない。
リージョンはグローバル座標で指定すること。
再配置が発生する。

Cの関数► int *WMAddRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A21A　　WMSubRect

引　数► long　　rectPtr　　　　; レクタングルレコードへのポインタ

返り値► D0.L　　リザルトコード

機　能► カレントウィンドウのアップデートリージョンから rectPtr で指定したレクタングルを除く。このコールを呼ぶのは$A20D WMUpdate の前でなければならない。
レクタングルはグローバル座標で指定すること。
再配置が発生する。

Cの関数► int *WMSubRect*(Rect *rectPtr);
返り値はリザルトコード。

$A21B　　WMSubRgn

引　数► long　　rgnHdl　　　　　; リージョンレコードへのハンドル

返り値► D0.L　　リザルトコード

機　能► カレントウィンドウのアップデートリージョンから rgnHdl で指定したリージョンを除く。このコールを呼ぶのは$A20D WMUpdate の前でなければならない。
リージョンはグローバル座標で指定すること。

再配置が発生する。

Cの関数► int *WMSubRgn*(Region **rgnHdl);
返り値はリザルトコード。

$A21C　WMGScriptSet

引　数► long　scriptHdl　;スクリプトへのハンドル
返り値► D0.L　リザルトコード
機　能► カレントウィンドウに、scriptHdl で指定したウィンドウスクリプトを設定する。
Cの関数► int *WMGScriptSet*(GScript **scriptHdl);
返り値はリザルトコード。

$A21D　WMGScriptGet

引　数► なし
返り値► A0.L　ウィンドウスクリプトへのハンドル
機　能► カレントウィンドウに設定されているウィンドウスクリプトへのハンドルを返す。

$A21E　WMTitleSet

引　数► long　winPtr　;ウィンドウレコードへのポインタ
long　strLPtr　;LASCII 型の文字列へのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウのタイトルを strLPtr で指定した文字列に変更し、ウィンドウを描き直す。疑似ポインタも可。
再配置が発生する。
Cの関数► int *WMTitleSet*(Window *winPtr, const _LASCII strLPtr);
返り値はリザルトコード。

$A21F　WMTitleGet

引　数► long　winPtr　;ウィンドウレコードへのポインタ
long　strLPtr　;LASCII 型の文字列バッファへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウのタイトルを、strLPtr で指定した文字列バッファに返す。
Cの関数► int *WMTitleGet*(Window *winPtr, _LASCII strLPtr);
結果は、LASCII 型のポインタ strLPtr で示されるバッファに格納される。

返り値はリザルトコード。

$A220　WMTIDSet

引　数► long　taskID　；タスク ID
返り値► D0.L　リザルトコード
機　能► カレントウィンドウのタスク ID を taskID に変更する。
Cの関数► int *WMTIDSet*(long taskID);
返り値はリザルトコード。

$A221　WMTIDGet

引　数► なし
返り値► D0.L　タスク ID
A0.L　ウィンドウレコードへのポインタ
機　能► カレントウィンドウのタスク ID を返す。
Cの関数► int *WMTIDGet*(void);
返り値はタスク ID。

$A222　WMPinRect　3.0

引　数► long　rectPtr　；レクタングルレコードへのポインタ
long　pt　；ポイント
返り値► D0.L　pt に最も近いポイント
機　能► pt が rectPtr で示された矩形領域の内側にあれば pt そのものを、外側にあれば矩形領域のなかで pt に最も近いポイントを返す。
Cの関数► LPoint *WMPinRect*(Rect *rectPtr, LPoint pt);
返り値はポイント。

$A223　WMCalcUpdt　3.0

引　数► long　rgnHdl　；リージョンレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► カレントウィンドウのアップデートリージョンに rgnHdl で指定したリージョンを加える。ウィンドウの外形も考慮される。このコールを呼ぶのは$A20D WMUpdate の前でなければならない。
リージョンはグローバル座標で指定すること。
再配置が発生する。

Cの関数► int *WMCalcUpdt*(Region **rgnHdl);
返り値はリザルトコード。

$A224　　WMGetDTGS

引　数► なし
返り値► A0.L　　グローバル変数 DeskTopGScript のアドレス
機　能► グローバル変数 DeskTopGScript のアドレスを返す。この変数には、背景の絵を描画するスクリプトへのハンドルが入っている。
Cの関数► GScript ****WMGetDTGS*(void);
返り値は DeskTopGScript のアドレス。

$A225　　WMDragRgn

引　数►

long	rgnHdl	; リージョンレコードのハンドル
long	pt	; ポイント（グローバル座標）
long	lRectPtr	; レクタングルレコードへのポインタ（グローバル座標）
long	sRectPtr	; レクタングルレコードへのポインタ（グローバル座標）
word	axis	; 移動制限フラグ

bit0	x 方向に	0:移動不可 1:移動可
bit1	y 方向に	0:移動不可 1:移動可

long	procPtr	; ユーザ定義の関数のアドレス

返り値► D0.L　　pt からの移動量を示すポイント/リザルトコード
機　能► マウスの左ボタンが押されている間、マウスの動きにあわせて rgnHdl で指定したリージョンの輪郭を移動させる。左ボタンが離された時点で終了する。
通常、pt としてはマウスが押された座標を指定する。
lRectPtr で示したレクタングルの範囲を超えるとリージョンの輪郭は移動しなくなり、sRectPtr で示したレクタングルの範囲を超えると輪郭は消える。
procPtr には、マウスの左ボタンが押されている間、繰り返し実行したい関数のアドレスを指定する。不要な場合は 0 を指定する。関数には、D6 レジスタで pt からの移動量を示すポイントが渡される。全レジスタを保存する必要がある。
再配置が発生する。
Cの関数► LPoint *WMDragRgn*(Region **rgnHdl, LPoint pt,
Rect *lRectPtr, Rect *sRectPtr, int axis,
void (*procPtr)());

返り値は pt からの移動量を意味するポイント。
負の値の場合はエラー。

$A22C　WMOptionGet

引　数► なし
返り値► D0.L　ウィンドウオプション（下位ワードのみ意味を持つ）
機　能► カレントウィンドウのウィンドウオプション（wOption）を返す。
Cの関数► int *WMOptionGet*(void);
返り値はウィンドウオプション。

$A22D　WMOptionSet

引　数► word　wOpt　；ウィンドウオプション
返り値► なし
機　能► カレントウィンドウのウィンドウオプション（wOption）を設定する。
Cの関数► void *WMOptionSet*(int wOpt);
返り値はない。

$A22E　WMPtInGBox

引　数► long　pt　；ローカル座標を意味するポイント
返り値► D0.L　= 0　サイズボタンのなかにはない
≠ 0　サイズボタンのなかにある
機　能► pt で指定したポイントがカレントウィンドウのサイズボタンの中にあるかどうかを調べる。
Cの関数► BOOLEAN *WMPtInGBox*(LPoint pt);
返り値は結果を意味する数値。

$A22F　WHOpen　3.0

引　数► long　pWinPtr　；親ウィンドウレコードへのポインタ
long　winPtr　；ウィンドウレコードへのポインタ
long　boundRectPtr　；レクタングルレコードへのポインタ
long　titleLStrPtr　；LASCII 型の文字列へのポインタ
long　visible　；可視フラグ
long　wDefID　；リソース 'WDEF' の ID
long　wOption　；ウィンドウオプション

ウィンドウマン

bit0	スクロールバー
bit1	クリップ
bit2	ドライブ表示
bit3	サイズボタン

long　behindWinPtr　；ウィンドウレコードへのポインタ
long　cBttn　；クローズボタンフラグ
long　taskID　；タスク ID 番号

返り値► D0.L　= 0　エラー
　≠ 0　ウィンドウレコードへのポインタ

機　能► pWinPtr で指定したウィンドウを親ウィンドウとするウィンドウを開く。pWinPtr として 0 を指定すると、ルートウィンドウ（＝デスクトップ画面）上に開く。
基本的に $A1F9 WMOpen と同様だが、wDefID を 16 倍する必要がないことと、wOption を別に指定する必要があること、そして返り値の戻し方が異なる点に注意。
再配置が発生する。

Cの関数► Window *_WHOpen_(HWindow *pWinPtr, Window *winPtr, Rect *boundRectPtr, LASCII *titleLStrPtr, int visible, int wDefID, int wOption, Window *behindWinPtr, int cBttn, long taskID);
返り値はウィンドウレコードへのポインタ。
エラーが発生した場合、NULL が返る。

$A230　WHGet　3.0

引　数► なし

返り値► D0.L　ルートウィンドウのウィンドウレコードへのポインタ
A0.L　ルートウィンドウのウィンドウレコードへのポインタ

機　能► ルートウィンドウのウィンドウレコードへのポインタを返す。

Cの関数► HWindow *_WHGet_(void);
返り値はウィンドウレコードへのポインタ。

$A231　WMOpen2　3.0

引　数► long　winPtr　；ウィンドウレコードへのポインタ
long　boundRectPtr　；レクタングルレコードへのポインタ
long　titleLStrPtr　；LASCII 型の文字列へのポインタ
long　visible　；可視フラグ
long　wDefID　；リソース‘WDEF’の ID
long　wOption　；ウィンドウオプション

ウィンドウマン

bit0	スクロールバー
bit1	クリップ
bit2	ドライブ表示
bit3	サイズボタン

long behindWinPtr ;ウィンドウレコードへのポインタ
long cBttn ;クローズボタンフラグ
long taskID ;タスクID番号

返り値► D0.L = 0 エラー
≠ 0 ウィンドウレコードへのポインタ

機　能► ウィンドウを開く。
基本的に$A1F9 WMOpenと同様だが、wDefIDを16倍する必要がないことと、wOptionを別に指定する必要があること、そして返り値の戻し方が異なる点に注意。
再配置が発生する。

Cの関数► Window *WMOpen2*(Window *winPtr, Rect *boundsRectPtr, LASCII *titleLStrPtr, int visible, int wDefID, int wOption, Window *behindWinPtr, int cBttn, long taskID);
返り値はウィンドウレコードへのポインタ。
エラーが発生した場合 NULL が返る。

$A232　WMMargineGet　3.0

引　数► long rectPtr ;レクタングルレコードへのポインタ

返り値► なし

機　能► rectPtrで指定したレクタングルレコードに、ズーム時のウィンドウのディスプレイレクタングルに対する上下左右のマージンを返す。レクタングルのleft, top, right, bottomが左上右下のマージンを意味する。

Cの関数► void *WMMargineGet*(Rect *rectPtr);
結果は、Rect型の変数rectPtrに格納される。

$A233　WMMargineSet　3.0

引　数► long rectPtr ;レクタングルレコードへのポインタ

返り値► なし

機　能► ウィンドウズーム時の、ディスプレイレクタングルに対するウィンドウの上下左右のマージンをrectPtrで示すレクタングルで指定する。レクタングルのleft, top, right, bottomが左上右下のマージンを意味する。
マージンの値は各方向とも 0〜128。

Cの関数► void *WMMargineSet*(Rect *rectPtr);

返り値はない。

$A235　　WMMove2　　3.0

引　数► long　winPtr　; ウィンドウレコードへのポインタ
long　pt　; ウィンドウの位置を示すポイント（グローバル座標）
word　active　; アクティブフラグ

返り値► D0.L　リザルトコード

機　能► winPtr で指定したウィンドウを pt で指定した位置に移動する。その際、active が -1 ならウィンドウをアクティブにしてから移動させる。
指定したウィンドウが子ウィンドウを持つ場合でも、子ウィンドウは移動しない。
再配置が発生する。

Cの関数► int *WMMove2*(Window *winPtr, LPoint pt, int active);
返り値はリザルトコード。

サブウィンドウマン

#include <WINDOW.H>

$A227　WSOpen

引　数► long　sWinPtr　；サブウィンドウレコードへのポインタ
long　rgnHdl　；アウトサイドリージョンとなるリージョンへのハンドル
long　prio　；プライオリティ値

返り値► D0.L　リザルトコード
A0.L　サブウィンドウレコードのアドレス

機　能► 新しいサブウィンドウを開く。
sWinPtr が 0 の場合、サブウィンドウマネージャがヒープ上に作成する。
再配置が発生する。

Cの関数► Subwin **WSOpen*(Subwin *sWinPtr,Region **rgnHdl,unsigned long prio);
返り値はサブウィンドウレコードへのポインタ。
エラーの場合は NULL が返る。

$A228　WSClose

引　数► long　sWinPtr　；サブウィンドウレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► sWinPtr で指定したサブウィンドウを閉じ、サブウィンドウリストから削除する。
サブウィンドウレコードをヒープ上以外に作成していた場合に使用する。
再配置が発生する。

Cの関数► int *WSClose*(Subwin *sWinPtr);
返り値はリザルトコード。

$A229　WSDispose

引　数► long　sWinPtr　；サブウィンドウレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► sWinPtr で指定したサブウィンドウを閉じ、サブウィンドウリストから削除したあと、サブウィンドウレコードとして確保されていたブロックを廃棄する。
サブウィンドウレコードをヒープ上に作成していた場合に使用する。

再配置が発生する。

Cの関数► int *WSDispose*(Subwin *sWinPtr);

返り値はリザルトコード。

$A22A WSEnlist

引　数► long sWinPtr ；サブウィンドウレコードへのポインタ

返り値► D0.L リザルトコード

機　能► sWinPtr で指定したサブウィンドウをサブウィンドウリストに加える。

Cの関数► int *WSEnlist*(Subwin *sWinPtr);

返り値はリザルトコード。

$A22B WSDelist

引　数► long sWinPtr ；サブウィンドウレコードへのポインタ

返り値► D0.L リザルトコード

機　能► sWinPtr で指定したサブウィンドウをサブウィンドウリストから削除する。

Cの関数► int *WSDelist*(Subwin *sWinPtr);

返り値はリザルトコード。

メニューマン

#include <MENU.H>

$A266 MNInit

引　数► なし
返り値► なし
機　能► メニューマンを初期化する。
Cの関数► void *MNInit*(void);
返り値はない。

$A267 MNRefer

引　数► word　menuID　; リソース 'MENU' の ID
返り値► D0.L　リザルトコード
A0.L　メニューレコードへのハンドル
機　能► リソースタイプ 'MENU'、ID menuID として定義されているメニューテンプレートの内容に従ってメニューを定義する。
再配置が発生する。
Cの関数► Menu ***MNRefer*(short menuID);
返り値はメニューレコードへのハンドル。

$A268 MNSelect

引　数► long　menuHdl　; メニューレコードへのハンドル
long　pt　; 表示を開始するポイント（グローバル座標系）
返り値► D0.L　= 0　何も選択されなかった
≠ 0　下位ワード: メニューアイテム番号（1～）
上位ワード: 階層メニュー ID/リザルトコード（負の値）
機　能► pt で指定したポイントから menuHdl で指定したメニューレコードの内容に従ってメニューを描画し、そのうえでマウスの右ボタンが離された（選択された）メニューアイテムの番号を返す。
通常、pt にはマウスの右ボタンが押されたポイントを指定する。
階層メニューが選択された場合、返り値の上位ワードは階層メニューテンプレートのリソース ID を意味する。階層メニュー表示の 0.3 秒のタイムラグ中にマウスのボタンが離された場合、親アイテムの番号が返される。

再配置が発生する。

Cの関数► int *MNSelect*(Menu **menuHdl, LPoint pt);

返り値は下位ワードがメニューアイテムの番号、上位ワードが階層メニューID。負の値の場合はリザルトコード。

$A269　MNConvert

引　数► long　menuHdl　；メニューレコードへのハンドル

long　strZPtr　；メニュー定義文字列（ASCIIZ）へのポインタ

word　ID　；メニュー定義関数のID

返り値► D0.L　リザルトコード

A0.L　メニューレコードへのハンドル

機　能► strZPtr で指定した文字列によって、メニューレコードを作成する。

menuHdl が 0 の場合、メニューマンがヒープ上に作成する。

メニュー定義文字列とは、基本的にメニューアイテムを1つずつカンマで区切ったもの。特殊文字を利用することによってショートカットやチェックマークの指定を行うことができる。

特殊文字	内　容
^	ショートカット文字の指定。次の1文字がショートカット文字となる
~	この文字で始まるアイテムはインアクティブとなる
!	チェックマークをつける

再配置が発生する。

Cの関数► Menu ****MNConvert**(Menu **menuHdl, const char *strZPtr, int ID);

返り値は、メニューレコードへのハンドルまたはリザルトコード。

$A26A　MNSelect2　2.0

引　数► long　menuHdl　；メニューレコードへのハンドル

long　pt　；表示を開始するポイント（グローバル座標系）

long　rectPtr　；レクタングルレコードへのポインタ（グローバル座標系）

返り値► D0.L　= 0　何も選択されなかった

≠ 0　下位ワード：メニューアイテム番号（1～）

上位ワード：階層メニューID/リザルトコード（負の値）

機　能► pt で指定したポイントから、menuHdl で指定したメニューレコードの内容に従ってメニューを描画し、そのうえでマウスの右ボタンが離された（選択された）メニューアイテムの番号を返す。

通常、pt にはマウスの右ボタンが押されたポイントを指定する。
メニューは、rectPtr で指定した範囲内に表示される。
階層メニューが選択された場合、返り値の上位ワードは階層メニューテンプレートのリソース ID を意味する。階層メニュー表示の 0.3 秒のタイムラグ中にマウスのボタンが離された場合、親アイテムの番号が返される。
再配置が発生する。

Cの関数► int *MNSelect2*(Menu **menuHdl, LPoint pt, Rect *rectPtr);
返り値は、下位ワードがメニューアイテムの番号、上位ワードが階層メニュー ID。
負の値の場合はリザルトコード。

$A26B　　MNConvert2　　3.0

引　数► long　　menuHdl　　; メニューレコードへのハンドル
　　　　long　　strZPtr　　; メニュー定義文字列（ASCIIZ）へのポインタ
　　　　word　　ID　　; メニュー定義関数の ID

返り値► D0.L　　リザルトコード
　　　　A0.L　　メニューレコードへのハンドル

機　能► strZPtr で指定した文字列によって、メニューレコードを作成する。
menuHdl が 0 の場合、メニューマンがヒープ上に作成する。
メニュー定義文字列とは、基本的にメニューアイテムを 1 つずつカンマで区切ったもの。特殊文字を利用することによって、ショートカットやチェックマークの指定を行うことができる。

<table>
<tr><th>特殊文字</th><th colspan="2">内容</th></tr>
<tr><td>^</td><td colspan="2">ショートカット文字の指定。次の 1 文字がショートカット文字となる</td></tr>
<tr><td>~</td><td colspan="2">この文字で始まるアイテムはインアクティブとなる</td></tr>
<tr><td>!</td><td colspan="2">チェックマークをつける</td></tr>
<tr><td>|</td><td colspan="2">次の 1 文字をチェックマークフラグの欄に格納する
エスケープキャラクタも使用可能（階層メニュー ID 指定用）</td></tr>
<tr><td>"</td><td colspan="2">読み飛ばされるので意味を持たない</td></tr>
<tr><td>\</td><td colspan="2">エスケープキャラクタ</td></tr>
<tr><td></td><td>\^</td><td>^</td></tr>
<tr><td></td><td>\~</td><td>~</td></tr>
<tr><td></td><td>\!</td><td>!</td></tr>
<tr><td></td><td>\|</td><td>|</td></tr>
<tr><td></td><td>\"</td><td>"</td></tr>
<tr><td></td><td>\\</td><td>\</td></tr>
<tr><td></td><td>\,</td><td>,</td></tr>
<tr><td></td><td>\ooo</td><td>ooo は 8 進数 3 桁以内</td></tr>
<tr><td></td><td>\xhh</td><td>hh は 16 進数 2 桁以内</td></tr>
</table>

再配置が発生する。

Cの関数► Menu ***MNConvert2*(Menu **menuHdl,char *strZPtr,short ID);

返り値はメニューレコードへのハンドル。
エラーが発生した場合、NULL が返る。

$A26C　MNSelect3　3.0

引　数► long　menuHdl　; メニューレコードへのハンドル
long　mousePt　; マウスの位置を意味するポイント（グローバル座標系）
long　menuPt　; メニューの表示開始位置を意味するポイント（グローバル座標系）
long　item　; 初期アイテム番号
long　rectPtr　; レクタングルレコードへのポインタ（グローバル座標系）

返り値► D0.L　= 0　何も選択されなかった
≠ 0　下位ワード：メニューアイテム番号（1～）
上位ワード：階層メニュー ID/リザルトコード（負の値）

機　能► menuHdl で指定したメニューレコードの内容に従ってメニューを描画し、そのうえでマウスの右ボタンが離された（選択された）メニューアイテムの番号を返す。メニューを描画する際、item で指定したアイテムが menuPt から表示されるように表示位置を調整する。mousePt には、ボタンが押されたときのマウスの座標を別途指定する。
メニューは、rectPtr で指定した範囲内に表示される。
階層メニューが選択された場合、返り値の上位ワードは階層メニューテンプレートのリソース ID を意味する。階層メニュー表示の 0.3 秒のタイムラグ中にマウスのボタンが離された場合、親アイテムの番号が返される。
ただし、システムに標準で付属するメニュー定義関数は、このコールに完全には対応していない。再配置が発生する。

Cの関数► int *MNSelect3*(Menu **menuHdl, LPoint mousePt,
LPoint menuPt, short *item, Rect *rectPtr);
返り値は下位ワードがメニューアイテムの番号、上位ワードが階層メニュー ID。負の値の場合はリザルトコード。

コントロールマン

#include <CONTROL.H>

$A289 CMOpen

引　数► long winPtr ；ウィンドウレコードへのポインタ
long boundsRectPtr ；レクタングルレコードへのポインタ
long titleLStrPtr ；タイトルを示す LASCII 型の文字列へのポインタ
word visible ；可視フラグ
word value ；コントロールの初期値
word min ；コントロールの最小値
word max ；コントロールの最大値
word cDefID ；リソース 'CDEF' の ID×16
long user ；ユーザワークの初期値

返り値► D0.L リザルトコード
A0.L コントロールレコードへのハンドル

機　能► winPtr で指定したウィンドウ中に新しいコントロールを作成する。
boundsRectPtr は、コントロールの存在する領域を指定するレクタングルレコードへのポインタ（疑似ポインタ可）。ローカル座標で指定する。
titleLStrPtr はコントロールのタイトルを指定する LASCII 型の文字列だが、コントロールの種類によっては無視される（疑似ポインタ可）。
visible として-1 を指定すると可視のコントロール、0 を指定すると不可視のコントロールとなる。ただし、$A28E CMDraw, $A28F CMDrawOne などを実行するまでは、実際には描画されない。
再配置が発生する。

Cの関数► Control ***CMOpen*(Window *winPtr, Rect *boundsRectPtr, const _LASCII *titleLStrPtr, BOOLEAN visible, int value, int min, int max, int cDefID, long user);
返り値はコントロールレコードへのハンドル。
エラーの場合は NULL が返る。

$A28A CMDispose

引　数► long ctrlHdl ；コントロールレコードへのハンドル

返り値► D0.L リザルトコード

機　能► ctrlHdl で指定したコントロールのレコードを廃棄し、コントロールリストか

ら外し、表示を消す。
再配置が発生する。
Cの関数► int *CMDispose*(Control **ctrlHdl);
返り値はリザルトコード。

$A28B　CMKill

引　数► long　winPtr　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウ上のコントロールをすべて廃棄する。
再配置が発生する。
Cの関数► int *CMKill*(Window *winPtr);
返り値はリザルトコード。

$A28C　CMHide

引　数► long　ctrlHdl　;コントロールレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► ctrlHdl で指定したコントロールを不可視にする。表示も消す。
再配置が発生する。
Cの関数► int *CMHide*(Control **ctrlHdl);
返り値はリザルトコード。

$A28D　CMShow

引　数► long　ctrlHdl　;コントロールレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► ctrlHdl で指定したコントロールを可視にする。再描画する。
再配置が発生する。
Cの関数► int *CMShow*(Control **ctrlHdl);
返り値はリザルトコード。

$A28E　CMDraw

引　数► long　winPtr　;ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウ上のコントロール（可視のもの）をすべて描画する。
再配置が発生する。

Cの関数► int *CMDraw*(Window *winPtr);
返り値はリザルトコード。

$A28F　CMDrawOne

引　数► long　ctrlHdl　;コントロールレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► ctrlHdl で指定したコントロール（可視のもの）を描画する。
再配置が発生する。
Cの関数► int *CMDrawOne*(Control **ctrlHdl);
返り値はリザルトコード。

$A290　CMValueSet

引　数► long　ctrlHdl　;コントロールレコードへのハンドル
word　value　;コントロールの値
返り値► D0.L　リザルトコード
機　能► ctrlHdl で指定したコントロールの値を value にする。変更した状態が再描画される。
再配置が発生する。
Cの関数► int *CMValueSet*(Control **ctrlHdl, int value);
返り値はリザルトコード。

$A291　CMValueGet

引　数► long　ctrlHdl　;コントロールレコードへのハンドル
返り値► D0.L　コントロールの値（下位ワード）
/リザルトコード（ロングワードの負の値）
機　能► ctrlHdl で指定したコントロールの値を返す。
Cの関数► int *CMValueGet*(Control **ctrlHdl);
返り値はリザルトコード。

$A292　CMMinSet

引　数► long　ctrlHdl　;コントロールレコードへのハンドル
word　min　;コントロールの最小値
返り値► D0.L　リザルトコード
機　能► ctrlHdl で指定したコントロールの最小値を min にする。変更した状態が再描画される。

再配置が発生する。

Cの関数► int *CMMinSet*(Control **ctrlHdl, int min);
返り値はリザルトコード。

$A293　CMMinGet

引　数► long　ctrlHdl　; コントロールレコードへのハンドル
返り値► D0.L　コントロールの最小値（下位ワード）
/リザルトコード（ロングワードの負の値）
機　能► ctrlHdl で指定したコントロールの最小値を返す。
Cの関数► int *CMMinGet*(Control **ctrlHdl);
返り値はリザルトコード。

$A294　CMMaxSet

引　数► long　ctrlHdl　; コントロールレコードへのハンドル
word　max　; コントロールの最大値
返り値► D0.L　リザルトコード
機　能► ctrlHdl で指定したコントロールの最大値を max にする。変更した状態が再描画される。
再配置が発生する。
Cの関数► int *CMMaxSet*(Control **ctrlHdl, int max);
返り値はリザルトコード。

$A295　CMMaxGet

引　数► long　ctrlHdl　; コントロールレコードへのハンドル
返り値► D0.L　コントロールの最大値（下位ワード）
/リザルトコード（ロングワードの負の値）
機　能► ctrlHdl で指定したコントロールの最大値を返す。
Cの関数► int *CMMaxGet*(Control **ctrlHdl);
返り値はリザルトコード。

$A296　CMMove

引　数► long　ctrlHdl　; コントロールレコードへのハンドル
long　pt　; コントロールを移動させるポイント（ローカル座標）
返り値► D0.L　リザルトコード

機　能► ctrlHdl で指定したコントロールを pt で指定したポイントに移動させ、再描画する。
再配置が発生する。

Cの関数► int *CMMove*(Control **ctrlHdl, LPoint pt);
返り値はリザルトコード。

$A297　　CMSize

引　数► long　ctrlHdl　　;コントロールレコードへのハンドル
long　sizePt　　;コントロールのサイズを示すポイント

返り値► D0.L　リザルトコード

機　能► ctrlHdl で指定したコントロールの大きさを、sizePt で指定した x, y のサイズに変更する。変更したサイズで再描画する。
再配置が発生する。

Cの関数► int *CMSize*(Control **ctrlHdl, LPoint sizePt);
返り値はリザルトコード。

$A298　　CMShine

引　数► long　ctrlHdl　　;コントロールレコードへのハンドル
word　partCode　　;パートコード

0	コントロール全体
1〜9	ユーザ用
10	標準ボタン
11	ボタン
12〜19	ユーザ用
20	アップボタン
21	ダウンボタン
22	アップページ
23	ダウンページ
24	タイトル
25〜128	ユーザ用
129	ドラッグできる部分
130〜254	RESERVED
255	インアクティブ状態

返り値► D0.L　リザルトコード

機　能► ctrlHdl で指定したコントロールの、partCode で指定した部分を強調表示する。0 を指定した場合はコントロール全体が強調され、255 を指定した場合はコントロール全体がインアクティブ状態で表示される。
再配置が発生する。

Cの関数► int *CMShine*(Control **ctrlHdl, int partCode);
返り値はリザルトコード。

$A299 CMFind

引 数► long pt ;ポイント（ローカル座標）
long winPtr ;ウィンドウレコードへのポインタ

返り値► D0.L = 0 該当するコントロールはない
≠ 0 パートコード（下位ワード）/リザルトコード
A0.L コントロールレコードへのハンドル

機 能► winPtr で指定したウィンドウのなかの pt で指定した座標にコントロールが存在するかどうかを調べ、存在した場合はコントロールレコードへのハンドルとパートコードを返す。

Cの関数► int *CMFind*(LPoint pt,Window *winPtr,Control ***ctrlHdl);
返り値はパートコード。
Control 型のポインタ変数 ctrlHdl に、該当するコントロールのコントロールレコードへのハンドルが返る。

$A29A CMCheck

引 数► long ctrlHdl ;コントロールレコードへのハンドル
long pt ;マウスのクリック座標（ローカル座標）
long procPtr ;ユーザ定義の関数のアドレス

返り値► D0.L パートコード（下位ワード）/リザルトコード

機 能► ctrlHdl で指定されたコントロールについて、マウスの左ボタンが離されるまで待ち、離されたポイントが指定したコントロール内であるかどうかを調べる。
マウスカーソルがコントロール上にある間は強調表示する。
通常、pt としてはマウスの左ボタンが押された座標、つまり、コントロール内のポイントを指定する。
procPtr には、マウスの左ボタンが押されている間、繰り返し実行したい関数のアドレスを指定する。-1 を指定した場合はそのコントロールのデフォルトの関数が使われ、0 を指定した場合は選択されたパートをハイライト表示するのみとなる。ユーザの指定した関数にはスタックを利用して以下の引数が渡される。
long ctrlHdl ;コントロールレコードへのハンドル
word cOption ;コントロールオプションコード
long pt ;移動量を示すポイント
再配置が発生する。

Cの関数► int *CMCheck*(Control **ctrlHdl, LPoint pt,
void (*procPtr)());
返り値は、パートコードまたはリザルトコード。

$A29B　CMRefer

引　数►	word	ctrlID	；リソース'CNTL'の ID×16
	long	winPtr	；ウィンドウレコードへのポインタ
返り値►	D0.L	リザルトコード	
	A0.L	コントロールレコードへのハンドル	

機　能► winPtrで指定したウィンドウのなかにリソースタイプ'CNTL'、ID ctrlIDとして定義されているコントロールテンプレートの内容に従って新しいコントロールを作成する。
再配置が発生する。

Cの関数► Control ***CMRefer*(int ctrlID, Window *winPtr);
返り値はコントロールレコードへのハンドル。
エラーの場合は NULL が返る。

$A29C　CMTitleGet

引　数►	long	ctrlHdl	；コントロールレコードへのハンドル
	long	strLPtr	；LASCII 型の文字列バッファへのポインタ
返り値►	D0.L	リザルトコード	

機　能► ctrlHdlで指定したコントロールのタイトルを、strLPtrで指定した文字列バッファに返す。
疑似ポインタも可。

Cの関数► int *CMTitleGet*(Control **ctrlHdl,const _LASCII *strLPtr);
返り値はリザルトコード。

$A29E　CMDraws

引　数►	long	winPtr	；ウィンドウレコードへのポインタ
	long	rgnHdl	；リージョンレコードへのハンドル
返り値►	D0.L	リザルトコード	

機　能► winPtrで指定したウィンドウ上のコントロール（可視のもの）のうち、rgnHdlで指定したリージョンの内部に一部でも含まれるものをすべて描画する。
再配置が発生する。

Cの関数► int *CMDraws*(Window *winPtr, Region **rgnHdl);
返り値はリザルトコード。

$A29F　CMTitleSet

引　数►	long	ctrlHdl	；コントロールレコードへのハンドル

long strLPtr ;LASCII 型の文字列へのポインタ

返り値► D0.L リザルトコード

機 能► ctrlHdl で指定したコントロールのタイトルを strLPtr で指定した文字列に変更し、再描画する。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *CMTitleSet*(Control **ctrlHdl,const _LASCII *strLPtr);
返り値はリザルトコード。

$A2A0 CMOptionGet

引 数► long ctrlHdl ;コントロールレコードへのハンドル

返り値► D0.L コントロールオプション(下位ワードのみ意味を持つ)

機 能► ctrlHdl で指定したコントロールのコントロールオプション(cOption)を返す。

Cの関数► unsigned short *CMOptionGet*(Control **ctrlHdl);
返り値はコントロールオプション。

$A2A1 CMOptionSet

引 数► long ctrlHdl ;コントロールレコードへのハンドル
word cOpt ;コントロールオプション

返り値► なし

機 能► ctrlHdl で指定したコントロールのコントロールオプション(cOption)を設定する。

Cの関数► void *CMOptionSet*(Control **ctrlHdl, int cOpt);
返り値はない。

$A2A2 CMUserGet

引 数► long ctrlHdl ;コントロールレコードへのハンドル

返り値► D0.L ユーザ用のワークの値

機 能► ctrlHdl で指定したコントロールのユーザ用のワーク(cUser)を返す。

Cの関数► long *CMUserGet*(Control **ctrlHdl);
返り値はユーザ用ワークの値。

$A2A3 CMUserSet

引 数► long ctrlHdl ;コントロールレコードへのハンドル

long　cUser　；ユーザ用のワークの値

返り値► なし

機　能► ctrlHdl で指定したコントロールのユーザ用のワーク (cUser) を設定する。

Cの関数► void *CMUserSet*(Control **ctrlHdl, long cUser);
返り値はない。

$A2A4　CMProcGet

引　数► long　ctrlHdl　；コントロールレコードへのハンドル

返り値► D0.L　ドラッグ時の手続きのアドレス

機　能► ctrlHdl で指定したコントロールのドラッグ時の手続きのアドレス (cProc) を返す。

$A2A5　CMProcSet

引　数► long　ctrlHdl　；コントロールレコードへのハンドル
long　cProc　；ドラッグ時の手続きのアドレス

返り値► なし

機　能► ctrlHdl で指定したコントロールのドラッグ時の手続きのアドレス (cProc) を設定する。

$A2A6　CMDefDataGet

引　数► long　ctrlHdl　；コントロールレコードへのハンドル

返り値► D0.L　定義関数のデータ

機　能► ctrlHdl で指定したコントロールの定義関数のデータ (cDefData) を返す。

Cの関数► long *CMDefDataGet*(Control **ctrlHdl);
返り値は定義関数のデータ。

$A2A7　CMDefDataSet

引　数► long　ctrlHdl　；コントロールレコードへのハンドル
long　cDefData　；定義関数のデータ

返り値► なし

機　能► ctrlHdl で指定したコントロールの定義関数のデータ (cDefData) を設定する。

Cの関数► void *CMDefDataSet*(Control **ctrlHdl, long cDefData);
返り値はない。

ダイアログマン

#include <DIALOG.H>

$A2C0　DMInit

引　数► なし

返り値► なし

機　能► ダイアログマンを初期化する。ウィンドウマンを初期化したあとに呼ぶこと。
再配置が発生する。

Cの関数► void *DMInit*(void);
返り値はない。

$A2C2　DMFontSet

引　数► word　fontKind　；フォントカインド

返り値► なし

機　能► ダイアログ中で使われるフォントカインドを指定する。デフォルトは0（ROM12フォント）。
この設定はグローバルであり、以降、開かれるダイアログすべてに影響する。さらに、画面状態を保存するモードでは、この設定も保存されるので、注意が必要である。
再配置が発生する。

Cの関数► void *DMFontSet*(int fontKind);
返り値はない。

$A2C3　DMOpen

引　数►

long	dlgPtr	；ダイアログレコードへのポインタ
long	boundRectPtr	；レクタングルレコードへのポインタ
long	titleLStrPtr	；LASCII 型の文字列へのポインタ
word	visible	；可視フラグ
word	wDefID	；リソース 'WDEF' の ID×16 ＋ オプション

オプション	
bit0	スクロールバー
bit1	クリップ
bit2	ドライブ表示
bit3	サイズボタン

	long	behindWinPtr	; ウィンドウレコードへのポインタ
	word	cBox	; クローズボタンフラグ
	long	taskID	; タスク ID
	long	itemHdl	; ダイアログアイテムリストへのハンドル
返り値►	D0.L	リザルトコード	
	A0.L	ダイアログレコードへのポインタ	

機　能► 新しいダイアログを開く。

dlgPtr は、新しく作成するダイアログのダイアログレコードが納められる場所を示すポインタ。0 を指定すると、ダイアログマンがヒープ上に作成する。

titleLStrPtr は、ダイアログタイトルを示す LASCII 型の文字列へのポインタ。

visible として-1 を指定するとダイアログは可視となり、0 を指定すると不可視となる。

behindWinPtr は、このダイアログの上に表示されるウィンドウのレコードへのポインタを指定する。-1 を指定すると最も手前に、0 を指定すると最も奥に置かれる。通常は-1 を指定する。

cBox として-1 を指定すると、クローズボタンを描画する。0 の場合は描画しない。通常は 0 を指定する。

再配置が発生する。

Cの関数►
```
Dialog *DMOpen(Dialog *dlgPtr, Rect *boundRectPtr,
const _LASCII titleLStrPtr, BOOLEAN visible,
int wDefID, Window *behindWinPtr, BOOLEAN cBox,
long taskID, _Handle **itemHdl);
```
返り値はダイアログレコードへのポインタ。

エラーの場合は NULL が返る。

$A2C4　　DMRefer

引　数►	word	dialogID	; リソース 'DLOG' の ID
	long	dlgPtr	; ダイアログレコードへのポインタ
	long	behindWinPtr	; ウィンドウレコードへのポインタ
返り値►	D0.L	リザルトコード	
	A0.L	ダイアログレコードへのポインタ	

機　能► リソースタイプ 'DLOG'、ID dialogID として定義されているダイアログテンプレートの内容に従って新しいダイアログを作成する。

dlgPtr は、新しく作成するダイアログのダイアログレコードが納められる場所を示すポインタ。0 を指定すると、ダイアログマンがヒープ上に作成する。

behindWinPtr は、このダイアログの上に表示されるウィンドウのレコードへのポインタを指定する。-1 を指定すると最も手前に、0 を指定すると最も奥に置かれる。通常は-1 を指定する。

再配置が発生する。

Cの関数► Dialog **DMRefer*(int dialogID, Dialog *dlgPtr, Window *behindWinPtr);
返り値はダイアログレコードへのポインタ。
エラーの場合は NULL が返る。

$A2C5 DMClose

引　数► long　dlgPtr　;ダイアログレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► dlgPtr で指定したダイアログを閉じ、ウィンドウリストから削除する。
ダイアログレコードをヒープ上以外に作成していた場合に使用する。
再配置が発生する。
Cの関数► int *DMClose*(Dialog *dlgPtr);

$A2C6 DMDispose

引　数► long　dlgPtr　;ダイアログレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► dlgPtr で指定したダイアログを閉じ、ウィンドウリストから削除する。
ダイアログレコードをヒープ上に作成していた場合に使用する。
再配置が発生する。
Cの関数► int *DMDispose*(Dialog *dlgPtr);

$A2C7 DMControl

引　数► long　procPtr　;フィルタ関数のアドレス
返り値► D0.L　アイテム番号/リザルトコード
機　能► カレントのダイアログ中の、帰還属性を持つアイテムが左クリックされるまで、アイテムの操作を行う。左クリックしたアイテムを強調し、その上でマウスボタンが離された場合、そのアイテム番号を返す。アイテムがスクロールバー等の場合はドラッグを終了した時点でアイテム番号を返す。
procPtr は、DMControl 実行中に受け取ったイベントを操作するためのフィルタ関数のアドレス。必要がない場合は 0 を指定する。フィルタ関数には次の引数がスタック経由で渡される。

long　dlgPtr　;ダイアログレコードへのポインタ
long　eventRec　;イベントレコードへのポインタ

結果はイベントレコードに返す。
再配置が発生する。

Cの関数► int *DMControl*(int (*procPtr)(Dialog *dlgPtr, Event *eventRec));
返り値はアイテム番号またはリザルトコード。

$A2C8　DMDraw

引　数► long　dlgPtr　; ダイアログレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► dlgPtr で指定したダイアログ内のすべてのアイテムを描画する。
再配置が発生する。
Cの関数► int *DMDraw*(Dialog *dlgPtr);
返り値はリザルトコード。

$A2CF　DIGet

引　数► long　dlgPtr　; ダイアログレコードへのポインタ
word　itemNo　; アイテム番号
long　typeBufPtr　; アイテムタイプが返るバッファへのポインタ
long　hdlBufPtr　; アイテムレコードのハンドルが返るバッファへのポインタ
long　rectBufPtr　; アイテムのレクタングルが返るバッファへのポインタ
返り値► D0.L　リザルトコード
機　能► dlgPtr で指定したダイアログ中の、アイテム番号 itemNo で示されるアイテムに関する情報を typeBufPtr, hdlBufPtr, rectBufPtr で示されるバッファに返す。
typeBufPtr は 1 ワード、hdlBufPtr は 1 ロングワード、rectBufPtr は 2 ロングワードのワークのアドレスを指していること。
再配置が発生する。
Cの関数► int *DIGet*(Dialog *dlgPtr, int itemNo, short *typeBufPtr, Handle *hdlBufPtr, Rect *rectBufPtr);
返り値はリザルトコード。

$A2D0　DISet

引　数► long　dlgPtr　; ダイアログレコードへのポインタ
word　itemNo　; アイテム番号

	word	itemType	;アイテムタイプ
	long	itemHdl	;アイテムレコードへのハンドル
	long	rectPtr	;アイテムの大きさを示すレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► dlgPtr で指定したダイアログ中のアイテム番号 itemNo で示されるアイテムに、itemType, itemHdl, rectPtr で示される情報を設定する。
コントロールマンのアイテムの場合、コントロールのバウンドレクタングルは影響を受けない。
再配置が発生する。

Cの関数► int *DISet*(Dialog *dlgPtr, int itemNo, short itemType, Handle itemHdl, Rect *rectPtr);
返り値はリザルトコード。

$A2D1　DITGet

引　数►	word	itemType	;アイテムタイプ
	long	itemHdl	;アイテムレコードへのハンドル
	long	strLPtr	;LASCII 型の文字列バッファへのポインタ

返り値► D0.L　リザルトコード

機　能► itemType, itemHdl で指定したアイテムのタイトルを、strLPtr で指定した文字列バッファに返す。itemHdl は、$A2CF DIGet で返されたものを使う。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *DITGet*(short itemType, Handle itemHdl, _LASCII strLPtr);
返り値はリザルトコード。

$A2D2　DITSet

引　数►	word	itemType	;アイテムタイプ
	long	itemHdl	;アイテムレコードのハンドル
	long	strLPtr	;LASCII 型の文字列バッファへのポインタ

返り値► D0.L　リザルトコード

機　能► itemType, itemHdl で指定したアイテムのタイトルを、strLPtr で指定した文字列に変更する。itemHdl は、$A2CF DIGet で返されたものを使う。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *DITSet*(short itemType, Handle itemHdl,

const _LASCII strLPtr);
返り値はリザルトコード。

$A2D3 DITSelect

引 数► long dlgPtr ;ダイアログレコードへのポインタ
word itemNo ;アイテム番号
long start ;選択開始オフセット
long end ;選択終了オフセット
返り値► D0.L リザルトコード
機 能► dlgPtrで指定したダイアログ中の、itemNoで指定したアイテム（編集可能テキストのみ）の、start文字からend文字までを選択状態にする。endとして $7FFFFFFF を指定すると、文字列の最後までが選択状態になる。
再配置が発生する。
Cの関数► int *DITSelect*(Dialog *dlgPtr, int itemNo, long start, long end);
返り値はリザルトコード。

$A2D6 DIUpdate

引 数► long dlgPtr ;ダイアログレコードへのポインタ
long rgnHdl ;リージョンレコードへのハンドル
返り値► D0.L リザルトコード
機 能► dlgPtrで指定したダイアログ中の、rgnHdlで示されたリージョンに一部でも含まれるアイテムをすべて再描画する。
再配置が発生する。
Cの関数► int *DIUpdate*(Dialog *dlgPtr, Region **rgnHdl);
返り値はリザルトコード。

$A2D7 DMBeep

引 数► word count ;BEEPを鳴らす回数
返り値► D0.L リザルトコード
機 能► BEEP音を（count -1）回鳴らす。
Cの関数► int *DMBeep*(int count);
返り値はリザルトコード。

$A2D8　　DIHide

引　数► long　dlgPtr　　　　; ダイアログレコードへのポインタ
　　　　 word　itemNo　　　　; アイテム番号
返り値► D0.L　リザルトコード
機　能► dlgPtr で指定したダイアログ中の、アイテム番号 itemNo で示されるアイテムを不可視にして表示を消去する。
再配置が発生する。
Cの関数► int *DIHide*(Dialog *dlgPtr, int itemNo);
返り値はリザルトコード。

$A2D9　　DIShow

引　数► long　dlgPtr　　　　; ダイアログレコードへのポインタ
　　　　 word　itemNo　　　　; アイテム番号
返り値► D0.L　リザルトコード
機　能► dlgPtr で指定したダイアログ中の、アイテム番号 itemNo で示されるアイテムを可視にして再表示する。
再配置が発生する。
Cの関数► int *DIShow*(Dialog *dlgPtr, int itemNo);
返り値はリザルトコード。

$A2F6　　DMError

引　数► word　flags　　　　; ダイアログの形式を指定するフラグ

上位バイト	パターンモード
$00	黄フラッグ
$01	赤フラッグ
$80	F1 クラッシュアニメーション
下位バイト	**ボタンモード**
$01	確認
$04	はい/いいえ
$05	登録/終了
$06	実行/取り消し
$07	継続/中止

　　　　 long　strZPtr　　　; エラー/警告メッセージ
返り値► D0.L　アイテム番号/リザルトコード
機　能► flags で指定した形式の簡易エラーメッセージ用ダイアログを作成し、その内部に strZPtr で指定された文字列を表示する。アイテムが左クリックされるのを待ち、アイテム番号を返す。
文字列中に適宜 CR コード ($0D) を入れることで改行する。

再配置が発生する。

Cの関数► int *DMError*(int flags, const char *strZPtr);
返り値はアイテム番号、またはリザルトコード。

$A2F7　DMWaitOpen

引　数► なし
返り値► D0.L　リザルトコード
A0.L　ダイアログレコードへのポインタ
機　能► ウェイトダイアログ（“しばらくお待ち下さい”ダイアログ）を表示する。
再配置が発生する。
Cの関数► int *DMWaitOpen*(void);
返り値はリザルトコード。

$A2F8　DMWaitClose

引　数► なし
返り値► D0.L　リザルトコード
機　能► ウェイトダイアログを消去する。
再配置が発生する。
Cの関数► int *DMWaitClose*(void);
返り値はリザルトコード。

$A2F9　DMWaitWhile

引　数► なし
返り値► D0.L　リザルトコード
A0.L　ダイアログレコードへのポインタ
機　能► ウェイトダイアログの踏切の絵を描き直す。前回の描き直しから1秒以上経過していた場合、次のパターンに描き換える。
ウェイトダイアログ表示中は適当な間隔を置いて呼び続けること。
再配置が発生する。
Cの関数► int *DMWaitWhile*(void);
返り値はリザルトコード。

$A2FA　DMError2　3.0

引　数► word　flags　；ダイアログの形式を指定するフラグ

上位バイト	パターンモード
$00	黄フラッグ
$01	赤フラッグ
$02	F1 クラッシュアニメーション
$04	ユーザ定義
$80	黒フラッグ
下位バイト	**ボタンモード**
$00	ユーザ定義
$01	確認
$04	はい/いいえ
$05	登録/終了
$06	実行/取り消し
$07	継続/中止


```
long    strZPtr        ; エラー/警告メッセージ
long    patTblPtr      ; パターンテーブルへのポインタ
long    bttnTblPtr     ; ボタンテーブルへのポインタ
long    pcmLength      ; ADPCM データのバイト数
long    pcmDataPtr     ; ADPCM データへのポインタ
```

返り値► D0.L アイテム番号/リザルトコード

機　能► flags で指定した形式の簡易エラーメッセージ用ダイアログを作成し、その内部に strZPtr で指定された文字列を表示する。アイテムが左クリックされるのを待ち、アイテム番号を返す。

flags でユーザ定義のパターンやボタンを指定した場合、patTblPtr が示すパターンテーブルや bttnTblPtr が示すボタンテーブルが参照される。

パターンテーブルの構造は以下のとおり。

```
patTbl:
        dc.l    rectImgPtr1     ; 1 コマ目のレクタングルイメー
                                  ジへのポインタ
        dc.l    count1          ; 表示間隔 (1/100 秒単位)
                  ⋮
        dc.l      0             ; エンドマーク
```

パターンはテキストタイプ 3 ページのレクタングルイメージ。2 コマ以上指定した場合は、それぞれのコマの表示間隔に従ってアニメーションする。

ボタンテーブルの構造は以下のとおり。

```
bttnTbl:
        dc.l    bttnStrZPtr1    ; 1 個目のボタンの文字列へのポ
                                  インタ
        dc.l    flag1           ; = 0      ユーザ定義ボタン
                                ; = 1      デフォルトボタン
                  ⋮
        dc.l      0             ; エンドマーク
```

ボタンの文字列は、半角 10 文字までの ASCIIZ 型文字列。

pcmLength が 0 以外の場合、ダイアログ表示時に pcmDataPtr で指定され

た ADPCM データ（15kHz）を鳴らす。

再配置が発生する。

Cの関数► int *DMError2*(int flags,char *strZPtr,ErrPat *patTblPtr,
ErrBttn *bttnTblPtr, int pcmLength,
unsigned char *pcmDataPtr);

返り値はアイテム番号、またはリザルトコード。

テキストマン

#include <TEXT.H>

$A30A TMInit

引　数► なし
返り値► D0.L　リザルトコード
機　能► テキストマンを初期化する。
再配置が発生する。
Cの関数► int *TMInit*(void);
返り値はリザルトコード。

$A30C TMSetRect

引　数► long　tEHdl　　;テキストエディットレコードへのハンドル
long　destRectPtr　　;レクタングルレコードへのポインタ
long　viewRectPtr　　;レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットレコードに、destRectPtr, viewRectPtr で示されたレクタングルをセットし、再表示する。
再配置が発生する。
Cの関数► int *TMSetRect*(TEdit **tEHdl, Rect *destRectPtr, Rect *viewRectPtr);
返り値はリザルトコード。

$A30D TMChangText

引　数► long　tEHdl　　;テキストエディットレコードへのハンドル
long　textPtr　　;テキストへのポインタ
long　length　　;テキストのバイト数
返り値► D0.L　新しくセットされたテキストのバイト数
/リザルトコード
機　能► tEHdl で指定したテキストエディットレコードに textPtr と length で指定したテキストを編集用にセットし、再表示を行う。
疑似ポインタも可。
再配置が発生する。
Cの関数► long *TMChangeText*(TEdit **tEHdl, const char *textPtr,

long length);
返り値は、セットされたテキストのバイト数。

$A30E TMIdle

引　数► long tEHdl ; テキストエディットレコードへのハンドル
返り値► D0.L リザルトコード
機　能► tEHdl で指定されたテキストエディットレコードのカーソルの点滅処理を駆動する。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
再配置が発生する。
Cの関数► int *TMIdle*(TEdit **tEHdl);
返り値はリザルトコード。

$A311 TMCaret

引　数► long tEHdl ; テキストエディットレコードへのハンドル
word mode ; カーソルのモード

0	消す
1	表示する

返り値► D0.L リザルトコード
機　能► tEHdl で指定したテキストエディットレコードのカーソルの状態を mode に従って設定する。
再配置が発生する。
Cの関数► int *TMCaret*(TEdit **tEHdl, int mode);
返り値はリザルトコード。

$A312 TMDispose

引　数► long tEHdl ; テキストエディットレコードへのハンドル
返り値► D0.L リザルトコード
機　能► tEHdl で指定したテキストエディットレコードを廃棄する。編集テキストも廃棄される。
再配置が発生する。
Cの関数► int *TMDispose*(TEdit **tEHdl);
返り値はリザルトコード。

$A313 TMUpDate

引　数► long tEHdl ; テキストエディットレコードへのハンドル

	long	updtRectPtr	;レクタングルレコードへのポインタ（ローカル座標）

返り値► D0.L　リザルトコード

機　能► tEHdl で指定された、テキストエディットレコードの編集テキストの、updtRect Ptr で指定したレクタングルで示される範囲を再表示する。あらかじめ目的のグラフレコードをカレントにしておく必要がある。
再配置が発生する。

Cの関数► int *TMUpDate*(TEdit **tEHdl, Rect *updtRectPtr);
返り値はリザルトコード。

$A314　TMSetText

引　数►	long	tEHdl	;テキストエディットレコードへのハンドル
	long	textPtr	;テキストへのポインタ
	long	length	;テキストのバイト数

返り値► D0.L　リザルトコード

機　能► tEHdl で指定したテキストエディットレコードに、textPtr と length で指定したテキストを編集用にセットする。再表示は行わない。
疑似ポインタも可。
再配置が発生する。

Cの関数► int *TMSetText*(TEdit **tEHdl, const char *textPtr, long length);
返り値はリザルトコード。

$A315　TMGetText

引　数►	long	tEHdl	;テキストエディットレコードへのハンドル
	long	buffPtr	;編集テキストを返すバッファのポインタ
	long	lenMax	;バッファのサイズ

返り値► D0.L　編集テキストのバイト数
/リザルトコード
A0.L　バッファに返されたテキストの終端 ＋1

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストを、buffPtr で指定されたバッファに返す。返される編集テキストのバイト数が lenMax を超える場合はエラーとなる。編集テキストの終端には$00 が置かれていないので注意。
再配置が発生する。

Cの関数► long *TMGetText*(TEdit **tEHdl,char *textPtr,long lenMax);
返り値は、編集テキストのバイト数またはリザルトコード。

$A316　TMSetSelect

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
long　selStart　；セレクト範囲の先頭位置のオフセット
long　selEnd　；セレクト範囲の終端位置のオフセット
long　selOffset　；前のセレクト位置のオフセット

返り値► D0.L　リザルトコード

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域を selStart, selEnd で指定した領域に新たに設定し、再表示する。selOffset は、前のセレクト位置の開始位置あるいは終了位置を指定する。開始位置を変更する場合は前の開始位置を、終了位置を変更する場合は前の終了位置を指定する。
再配置が発生する。

Cの関数► int *TMSetSelect*(TEdit **tEHdl,long selStart,long selEnd, long selOffset);
返り値はリザルトコード。

$A317　TMKey

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
word　keyChar　；入力キャラクタ

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストに、keyChar で指定されたキャラクタを入力して再表示する。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
マルチフォントテキストエディットの場合、カレントのスタイル情報で入力される。
再配置が発生する。

Cの関数► int *TMKey*(TEdit **tEHdl, int keyChar);
返り値は、編集結果またはリザルトコード。

$A318　TMStr

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
long　textPtr　；テキストへのポインタ
long　length　；テキストのバイト数

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域を、textPtr で指定した文字列と置き換える。CR($0D) 以外の制御コードを含めることはできない。再表示は行わない。
マルチフォントテキストエディットの場合、カレントのスタイル情報で入力される。
再配置が発生する。

Cの関数► int *TMStr*(TEdit **tEHdl,const char *textPtr,long length);
返り値は、編集結果またはリザルトコード。

$A319　　TMCalText

引　数► long　　tEHdl　　　　　; テキストエディットレコードへのハンドル

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定したテキストエディットの段落情報を計算/設定する。カーソルの位置などは変更されないので、通常は$A464 TMSetSelCal を利用する。再表示は行われない。
再配置が発生する。

Cの関数► int *TMCalText*(TEdit **tEHdl);
返り値はリザルトコード。

$A31A　　TMPinScroll

引　数► long　　tEHdl　　　　　; テキストエディットレコードへのハンドル

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト部分がビューレクタングルに収まるようにスクロールさせる。実際にスクロールした場合は再表示を行う。
あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
再配置が発生する。

Cの関数► int *TMPinScroll*(TEdit **tEHdl);
返り値はリザルトコード。

$A31B　　TMClick

引　数► long　　tEHdl　　　　　; テキストエディットレコードへのハンドル
long　　pt　　　　　　; マウスが左クリックされたポイント（ローカル座標）

word　extend　　; 動作モード

0	新規セレクト領域
1	前回のセレクト領域を変更
2	セレクト領域開始位置の変更
3	セレクト領域終了位置の変更

返り値► D0.L　リザルトコード

機　能► tEHdl で指定されたテキストエディットレコードのセレクト領域を、マウスによって変更する。pt には、マウスが左クリックされた座標を入れる。
あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
再配置が発生する。
extend で指定した値によって動作が異なる。

extend が 0 の場合
　pt で指定した位置とマウスの左ボタンを離した位置が、セレクト領域の開始位置、終了位置となる。

extend が 1 の場合
　前回指定したセレクト領域の開始位置は固定で、その位置とマウスの左ボタンを離した位置が、セレクト領域の開始位置、終了位置となる。

extend が 2 の場合
　前回指定したセレクト領域の終了位置が新しいセレクト領域の終了位置に、マウスの左ボタンを離した位置が新しいセレクト領域の開始位置となる。

extend が 3 の場合
　前回指定したセレクト領域の開始位置が新しいセレクト領域の開始位置に、マウスの左ボタンを離した位置が新しいセレクト領域の終了位置となる。

Cの関数► int *TMClick*(TEdit **tEHdl, LPoint pt, int extend);
返り値はリザルトコード。

$A31C　TMEvent

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル
long　eventRecPtr　　; イベントレコードへのポインタ

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードについて、eventRecPtr で指定されたイベントレコードの内容に対応する処理を行う。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
再配置が発生する。
対応するのは、以下の 4 つのイベント。これらの処理のあと、再表示が行われ

る。

● ヌルイベント

　　カーソルの点滅を行う。

● マウスレフトダウンイベント

　　セレクト領域の変更を行う。

● マウスライトダウンイベント

　　ポップアップメニューを表示して、テキストマンスクラップとの間でカット&ペーストを行う。

● キーダウンイベント

　　キャラクタの入力を行う（全角にも対応）。カーソルキー、BS キー、DEL キーに対応する。

Cの関数► int *TMEvent*(TEdit **tEHdl, TsEvent *eventRecPtr);
返り値は、編集結果またはリザルトコード。

$A320　TMCut

引　数► long　tEHdl　；テキストエディットレコードへのハンドル

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域をカットし、テキストマンスクラップに移し、再表示する。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
マルチフォントテキストエディットの場合、スタイル情報もテキストマンスクラップに移される。
再配置が発生する。

Cの関数► int *TMCut*(TEdit **tEHdl);
返り値は、編集結果またはリザルトコード。

$A321　TMCopy

引　数► long　tEHdl　；テキストエディットレコードへのハンドル

返り値► D0.L　リザルトコード

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト部分をテキストマンスクラップにコピーし、再表示する。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
マルチフォントテキストエディットの場合、スタイル情報もテキストマンスクラップに移される。
再配置が発生する。

Cの関数► int *TMCopy*(TEdit **tEHdl);

返り値はリザルトコード。

$A322　　TMPaste

引　数► long　tEHdl　；テキストエディットレコードへのハンドル

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域をテキストマンスクラップの内容と置き換え、再表示する。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。テキストマンスクラップの内容は変化しない。
マルチフォントテキストエディットの場合、スタイル情報もペーストされる。
再配置が発生する。

Cの関数► int *TMPaste*(TEdit **tEHdl);
返り値は、編集結果またはリザルトコード。

$A323　　TMDelete

引　数► long　tEHdl　；テキストエディットレコードへのハンドル

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域を消去し、再表示する。テキストマンスクラップの内容は変化しない。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
再配置が発生する。

Cの関数► int *TMDelete*(TEdit **tEHdl);
返り値は、編集結果またはリザルトコード。

$A324　　TMInsert

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
long　strPtr　；文字列のアドレス
long　length　；文字列のバイト数

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定したテキストエディットのセレクト領域と strPtr で指定した

文字列を置き換えて再表示する。
マルチフォントテキストエディットの場合、カレントのスタイル情報で入力される。
再配置が発生する。

Cの関数► int *TMInsert*(TEdit **tEHdl, const char *strPtr, long length);
返り値は、編集結果またはリザルトコード。

$A325 TMFromScrap

引　数► なし
返り値► D0.L　リザルトコード
機　能► デスクトップスクラップの内容を、テキストマンスクラップにコピーする。
再配置が発生する。
Cの関数► int *TMFromScrap*(void);
返り値はリザルトコード。

$A326 TMToScrap

引　数► なし
返り値► D0.L　リザルトコード
機　能► テキストマンスクラップの内容を、デスクトップスクラップにコピーする。
再配置が発生する。
Cの関数► int *TMToScrap*(void);
返り値はリザルトコード。

$A327 TMScrapHandle

引　数► なし
返り値► D0.L　リザルトコード
A0.L　テキストマンスクラップへのハンドル
機　能► テキストマンスクラップのハンドルを返す。
Cの関数► int *TMScrapHandle*(Handle *scrapHdl);
結果はハンドル scrapHdl に格納される。
返り値はリザルトコード。

$A328 TMGetScrapLen

引　数► なし

返り値► D0.L　　テキストマンスクラップのバイト数
　　　　　　　/リザルトコード
機　能► テキストマンスクラップのバイト数を返す。
Cの関数► long *TMGetScrapLen*(void);
　　　　返り値は、テキストマンスクラップのバイト数またはリザルトコード。

$A32B　TMTextBox2

引　数► long　textPtr　; 文字列へのポインタ
　　　　long　length　; 文字列のバイト数
　　　　long　destRectPtr　; レクタングルレコードへのポインタ
　　　　word　mode　; 行揃えモード

0	左寄せ
1	センタリング
-1	右寄せ

　　　　word　lineHeight　; 改行幅のドット数
返り値► D0.L　リザルトコード
機　能► textPtr で指定した文字列を mode で指定した行揃えにし、destRectPtr で指定されたレクタングルに納まるように表示する。あらかじめ、目的のグラフレコードをカレントにしておく必要がある。
　　　　textPtr, destRectPtr は疑似ポインタも可。
　　　　再配置が発生する。
Cの関数► int *TMTextBox2*(const char *textPtr, long length, Rect *destRectPtr, int mode, int lineHeight);
　　　　返り値はリザルトコード。

$A32C　TMCacheON

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
　　　　long　size　; キャッシュのバイト数
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットレコードについて、size で指定したサイズのキャッシュを用意し、ON にする。
　　　　再配置が発生する。
Cの関数► int *TMCacheON*(TEdit **tEHdl, long size);
　　　　返り値はリザルトコード。

$A32D　TMCacheOFF

引　数► long　tEHdl　; テキストエディットレコードへのハンドル

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定したテキストエディットレコードのキャッシュを廃棄し、OFF にする。
再配置が発生する。

Cの関数► int *TMCacheOFF*(TEdit **tEHdl);
返り値はリザルトコード。

$A32E　　TMCacheFlush

引　数► long　　tEHdl　　　　; テキストエディットレコードへのハンドル

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定したテキストエディットレコードのキャッシュをフラッシュする。
再配置が発生する。

Cの関数► int *TMCacheFlush*(TEdit **tEHdl);
返り値はリザルトコード。

$A32F　　TMShow

引　数► long　　tEHdl　　　　; テキストエディットレコードへのハンドル

返り値► D0.L　　ドローレベルを＋1 した結果の値

機　能► tEHdl で指定したテキストエディットレコードのドローレベルを＋1 する。再描画は行わない。

Cの関数► int *TMShow*(TEdit **tEHdl);
返り値は処理後のドローレベル。

$A330　　TMHide

引　数► long　　tEHdl　　　　; テキストエディットレコードへのハンドル

返り値► D0.L　　ドローレベルを −1 した結果の値

機　能► tEHdl で指定したテキストエディットレコードのドローレベルを −1 する。再描画は行わない。

Cの関数► int *TMHide*(TEdit **tEHdl);
返り値は処理後のドローレベル。

$A331　　TMSelShow

引　数► long　　tEHdl　　　　; テキストエディットレコードへのハンドル

返り値► D0.L　　ハイライト表示レベルを＋1 した結果の値

機　能► tEHdl で指定したテキストエディットレコードのハイライト表示レベルを＋1 する。再描画は行わない。

Cの関数► int *TMSelShow*(TEdit **tEHdl);
返り値は処理後のハイライト表示レベル。

$A332　　TMSelHide

引　数► long　tEHdl　；テキストエディットレコードへのハンドル

返り値► D0.L　ハイライト表示レベルを −1 した結果の値

機　能► tEHdl で指定したテキストエディットレコードのハイライト表示レベルを −1 する。再描画は行わない。

Cの関数► int *TMSelHide*(TEdit **tEHdl);
返り値は処理後のハイライト表示レベル。

$A333　　TMSearchStrF

引　数►

long	tEHdl	；テキストエディットレコードへのハンドル
long	strPtr	；検索文字列へのポインタ
long	length	；検索文字列のバイト数
long	offset1	；テキストの検索開始位置（オフセット）
long	offset2	；テキストの検索終了位置（オフセット）
long	procPtr	；フィルタプロセスのアドレス（0 で指定しない）

返り値► D0.L　発見した文字列の位置（オフセット）
　=-1　発見できなかった
A0.L　発見した文字列のバイト数

機　能► tEHdl で指定したテキストエディットの offset1, offset2 で示される範囲で、strPtr で指定した文字列を前方検索する。大文字小文字等は区別される。
procPtr で指定するフィルタプロセスは、検索処理中、適当な間隔で呼び出される。引数として、A0 に tEHdl が渡される。返り値として D0 に 0 以外を返すと、検索は中断され、TMSearchStrF の返り値としてフィルタプロセスの返り値がアプリケーションに返される。
再配置が発生する。

Cの関数► int *TMSearchStrF*(TEdit **tEHdl, const char *strPtr,
long length, long offset1, long offset2,
int (*procPtr)(TEdit **tEHdl), long *size);
返り値は発見した文字列の位置（オフセット）。
int 型の変数 size に発見した文字列のバイト数が格納される。

$A334　TMSearchStrB

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　strPtr　; 検索文字列へのポインタ
long　length　; 検索文字列のバイト数
long　offset1　; テキストの検索開始位置（オフセット）
long　offset2　; テキストの検索終了位置（オフセット）
long　procPtr　; フィルタプロセスのアドレス（0 で指定しない）

返り値► D0.L　発見した文字列の位置（オフセット）
=-1　発見できなかった
A0.L　発見した文字列のバイト数

機　能► tEHdl で指定したテキストエディットの offset1, offset2 で示される範囲で、strPtr で指定した文字列を後方検索する。大文字小文字等は区別される。
procPtr で指定するフィルタプロセスは、検索処理中、適当な間隔で呼び出される。引数として、A0 に tEHdl が渡される。返り値として D0 に 0 以外を返すと検索は中断され、TMSearchStrB の返り値としてフィルタプロセスの返り値がアプリケーションに返される。
再配置が発生する。

Cの関数► int *TMSearchStrB*(TEdit **tEHdl, const char *strPtr, long length, long offset1, long offset2, int (*procPtr)(TEdit **tEHdl), long *size);
返り値は発見した文字列の位置（オフセット）。
int 型の変数 size に発見した文字列のバイト数が格納される。

$A335　TMTextInWidth2

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　strPtr　; 文字列へのポインタ
long　offset　; 文字列のオフセット
word　width　; 文字列を納める幅（ドット数）
word　startPoint　; 文字列のローカル座標―ディスティネーションレクタングルの水平方向オフセット

返り値► D0.L　納まる文字列のバイト数/リザルトコード
A0.L　= 0　改行コード以外の理由で終了した
≠ 0　改行コードにより終了した

機　能► tEHdl で指定したテキストエディットレコードの環境下で、startPoint で指定した位置から strPtr, offset で指定される文字列を描画する場合、width で指定したドット数の幅に納まる文字数を計算し、結果を返す。

$A197 GMStrLength とはコントロールコードの処理が異なる。$0A，$0D(改行コード) があった場合、そこまでで文字数の計算を終了し、$09 (TAB コード) があった場合、TAB サイズ (teTabSize) に従ってタブの処理を行う。また、他のコントロールコードの場合、編集モード (teDrawMode) に従って計算を行う。
マルチフォントテキストエディットの場合、offset には編集テキスト先頭からのオフセットを指定する必要がある。
再配置が発生する。

Cの関数► int *TMTextInWidth2*(TEdit **tEHdl, const char *strPtr, long offset,int width,int startPoint,BOOLEAN *endMode);
返り値は納まる文字列のバイト数。
BOOLEAN 型の変数 endMode には、終了した理由を意味する値が格納される。

$A336　TMTextWidth2

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル
long　strPtr　　; 文字列へのポインタ
long　offset　　; 文字列のオフセット
word　length　　; 文字列のバイト数
word　startPoint　　; 文字列のローカル座標—ディスティネーションレクタングルの水平方向オフセット

返り値► D0.L　文字列の占める幅 (ドット数)/リザルトコード

機　能► tEHdl で指定したテキストエディットレコードの環境下で、startPoint で指定した位置から strPtr, offset, length で指定される文字列を描画する場合、それが占める幅 (ドット数) を計算し、結果を返す。
$A196 GMStrWidth とはコントロールコードの処理が異なる。$0A，$0D(改行コード) があった場合、そこまででドット数の計算を終了し、$09(TAB コード) があった場合、TAB サイズ (teTabSize) に従ってタブの処理を行う。
また、他のコントロールコードの場合、編集モード (teDrawMode) に従って計算を行う。
マルチフォントテキストエディットの場合、offset には編集テキスト先頭からのオフセットを指定する必要がある。
再配置が発生する。

Cの関数► int *TMTextWidth2*(TEdit **tEHdl,const char *strPtr, long offet, int length, int startPoint);
返り値は、文字列の占める幅またはリザルトコード。

$A337　TMDrawText2

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル

	long	strPtr	; 文字列のアドレス
	long	offset	; 文字列のオフセット
	word	length	; 文字列のバイト数
	word	startPoint	; 文字列のローカル座標—ディスティネーションレクタングルの水平方向オフセット
	word	tabMode	= 0　タブはペン位置を移動するだけ ≠ 0　タブはバックグラウンドカラーによる塗りつぶし
返り値►	D0.L	リザルトコード	

機　能► tEHdl で指定したテキストエディットレコードの環境下で、startPoint で指定した位置から strPtr, offset, length で指定される文字列を描画する。あらかじめビューレクタングルでクリップ領域を設定しておく必要がある。

$A196 GMStrWidth とはコントロールコードの処理が異なる。$0A, $0D(改行コード）があった場合、そこまでで描画を終了し、$09(TAB コード） があった場合、TAB サイズ（teTabSize） に従ってタブの処理を行う。また、他のコントロールコードの場合、編集モード（teDrawMode）に従って描画する。

マルチフォントテキストエディットの場合、offset には編集テキスト先頭からのオフセットを指定する必要がある。

再配置が発生する。

Cの関数► int *TMDrawText2*(TEdit **tEHdl, const char *strPtr, long offset,int length,int startPoint,BOOLEAN tabMode);

返り値はリザルトコード。

$A338　　TMUpDate2

引　数►	long	tEHdl	; テキストエディットレコードへのハンドル
	long	hisRecPtr	; 編集履歴レコードへのポインタ
返り値►	D0.L	リザルトコード	

機　能► hisRecPtr で指定した編集履歴レコードに従って tEHdl で指定したテキストエディットの描画を行う。

再配置が発生する。

Cの関数► int *TMUpDate2*(TEdit **tEHdl, TEHis *hisRecPtr);

返り値はリザルトコード。

$A339　　TMUpDate3

引　数►	long	tEHdl	; テキストエディットレコードへのハンドル
	long	updtRectPtr	; レクタングルレコードへのポインタ（ローカル座標）

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定したテキストエディットの updtRectPtr で指定したレクタングルで示される範囲を再表示する。$A313 TMUpDate との違いは、バックグラウンドカラーによる塗りつぶしを行わない点。
再配置が発生する。

Cの関数► int *TMUpDate3*(TEdit **tEHdl, Rect *updtRectPtr);
返り値はリザルトコード。

$A33A　　TMCalCOLine

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル
long　columnPtr　　; 段落情報を作成するバッファ($18 バイト)へのポインタ
long　offset　　; 段落位置

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定したテキストエディットの、offset で指定した段落位置(段落の通し番号)の段落情報を columnPtr からのバッファに作成する。
再配置が発生する。

Cの関数► long *TMCalCOLine*(TEdit **tEHdl, TEColumn *columnPtr, long offset);
返り値はリザルトコード。

$A33C　　TMCalLine

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル
long　columnPtr　　; 段落情報を作成するバッファ($18 バイト)へのポインタ
long　offset　　; 行位置/バイト位置
word　mode

0	offset はバイト位置
1	offset は行位置

返り値► D0.L　　この段落のバイト位置/リザルトコード

機　能► tEHdl で指定したテキストエディットの、offset で指定した行位置(mode=1 の場合)、あるいはバイト位置(mode= 0 の場合)の段落情報を、columnPtr からのバッファに作成する。
再配置が発生する。

Cの関数► int *TMCalLine*(TEdit **tEHdl, TEColumn *columnPtr, long offset, BOOLEAN mode);
返り値は、段落のバイト位置またはリザルトコード。

$A33D　　TMLeftSel

引　数► long　　tEHdl　　　　　；テキストエディットレコードへのハンドル
返り値► D0.L　　オフセット
　　　　　　　=-1　セレクト領域がテキストの先頭
機　能► tEHdl で指定したテキストエディットのセレクト領域の直前（1つ左）の位置を返す。
Cの関数► long *TMLeftSel*(TEdit **tEHdl);
返り値はオフセット。

$A33E　　TMRightSel

引　数► long　　tEHdl　　　　　；テキストエディットレコードへのハンドル
返り値► D0.L　　オフセット
　　　　　　　=-1　セレクト領域がテキストの最後
機　能► tEHdl で指定したテキストエディットのセレクト領域の直後（1つ右）の位置を返す。
Cの関数► int *TMRightSel*(TEdit **tEHdl);
返り値はオフセット。

$A33F　　TMPointSel

引　数► long　　tEHdl　　　　　；テキストエディットレコードへのハンドル
　　　　long　　xpoint　　　　；水平座標（ローカル座標）
　　　　long　　ypoint　　　　；垂直座標（ローカル座標）
　　　　word　　extend

0	新規セレクト領域
1	前回のセレクト領域を変更
2	セレクト領域開始位置の変更
3	セレクト領域終了位置の変更

返り値► D0.L　　リザルトコード
機　能► tEHdl で指定されたテキストエディットのセレクト位置を xpoint, ypoint で指定した座標によって変更し、再表示する。セレクト領域変更後にカーソル位置の再計算を行うが、スクロールは行わない。
再配置が発生する。
Cの関数► int *TMPointSel*(TEdit **tEHdl, int xpoint, int ypoint, int extend);
返り値はリザルトコード。

$A340　TMOffsetSel

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　offsett　; バイト位置
word　extend

0	新規セレクト領域
1	前回のセレクト領域を変更
2	セレクト領域開始位置の変更
3	セレクト領域終了位置の変更

返り値► D0.L　リザルトコード

機　能► tEHdl で指定されたテキストエディットのセレクト位置を、offset で指定したバイト位置によって変更し、再表示する。セレクト領域変更後にカーソル位置の再計算を行うが、スクロールは行わない。
再配置が発生する。

Cの関数► int *TMOffsetSel*(TEdit **tEHdl, long offset, int extend);
返り値はリザルトコード。

$A341　TMPointToLine

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　xpoint　; 水平座標（ローカル座標）
long　ypoint　; 垂直座標（ローカル座標）
long　columnPtr　; 段落情報を作成するバッファ($18 バイト）のアドレス

返り値► D0.L　この段落のバイト位置/リザルトコード
A0.L　段落情報を作成するバッファへのポインタ

機　能► tEHdl で指定したテキストエディットの、xpoint, ypoint で指定した座標の段落情報を columnPtr からのバッファに作成する。
再配置が発生する。

Cの関数► long *TMPointToLine*(TEdit **tEHdl,int xpoint,int ypoint, TEColumn *columnPtr);
返り値は、段落のバイト位置またはリザルトコード。

$A343　TMCalSelPoint

引　数► long　tEHdl　; テキストエディットレコードへのハンドル

返り値► D0.L　リザルトコード

機　能► tEHdl で指定された、テキストエディットの現在のセレクト領域の行位置と座標を再計算する。
再配置が発生する。

Cの関数► int *TMCalSelPoint*(TEdit **tEHdl);

返り値はリザルトコード。

$A345　　TMSetView

引　数► long　tEHdl　　　; テキストエディットレコードへのハンドル
long　viewRectPtr　　; レクタングルレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットのビューレクタングルとして、viewRectPtr で指定したレクタングルを設定する。
再配置が発生する。
Cの関数► int *TMSetView*(TEdit **tEHdl, Rect *viewRectPtr);
返り値はリザルトコード。

$A346　　TMScroll

引　数► long　tEHdl　　　; テキストエディットレコードへのハンドル
long　dh　　　; 水平方向にスクロールさせるドット数
long　dv　　　; 垂直方向にスクロールさせるドット数
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットを dh, dv で指定したドットだけ横・縦にスクロールさせ、再表示する。dh は正で右、負で左方向へ、dv は正で下、負で上方向にスクロールする。
再配置が発生する。
Cの関数► int *TMScroll*(TEdit **tEHdl, int dh, int dv);
返り値はリザルトコード。

$A347　　TMPointScroll

引　数► long　tEHdl　　　; テキストエディットレコードへのハンドル
long　xpoint　　　; 水平座標（ローカル座標）
long　ypoint　　　; 垂直座標（ローカル座標）
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットのビューレクタングルに xpoint,ypoint で指定した座標が納まるようにスクロールさせ、再表示する。
再配置が発生する。
Cの関数► int *TMPointScroll*(TEdit **tEHdl, int xpoint, int ypoint);
返り値はリザルトコード。

$A348　　TMStr2

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　strPtr　; テキストへのポインタ
long　length　; テキストのバイト数
long　hisRecPtr　; 編集履歴レコードへのポインタ

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域を strPtr で指定した文字列と置き換え、編集履歴レコードを作成する。
カーソル位置の再計算は行われない。
再表示は行わない。
マルチフォントテキストエディットの場合、カレントのスタイル情報で入力される。
再配置が発生する。

Cの関数► int *TMStr2*(TEdit **tEHdl,const char *strPtr,long length, TEHis *hisRecPtr);
返り値は、編集結果またはリザルトコード。

$A349　　TMKeyToAsk

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　eventRecPtr　; イベントレコードへのポインタ

返り値► D0.L　リザルトコード

-1	キーコードが格納されていない
-2	該当するコードがファンクションキーアサインテーブルに登録されていない

機　能► eventRecPtr で指定したイベントレコードに格納されているキーコードを、tEHdl で指定したテキストエディットレコードのファンクションキーアサインテーブルによって変換し、イベントレコードに再設定する。

Cの関数► int *TMKeyToAsk*(TEdit **tEHdl, TsEvent *eventRecPtr);
返り値は、エラー内容またはリザルトコード。

$A34A　　TMNextCode

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
word　code　; キーコード
=0　すべてのコード
word　mode　=0　イベントを取り除かない

≠ 0　イベントを取り除く

返り値► D0.L　上位ワード：　キーコード、下位ワード：　ASCII コード
= 0　指定したコードのキーイベントは発見できなかった

機　能► 次のキーダウンイベントを先読みし、tEHdl で指定したテキストエディットレコードのキーアサインテーブルによってキーコードを変換したあと、code で指定したキーコードであるかどうかをチェックする。指定したコードであった場合、mode に 0 以外を指定すると、このイベントを取り除く。

Cの関数► long *TMNextCode*(TEdit **tEHdl, int code, BOOLEAN mode);
返り値は上位ワードがキーコード、下位ワードが ASCII コード。イベントが発見できなかった場合は 0 。

$A34B　TMSetTextH

引　数►

long	tEHdl	; テキストエディットレコードへのハンドル
long	textHdl	; テキストへのハンドル
long	length	; テキストのバイト数

返り値► D0.L　リザルトコード

機　能► tEHdl で指定したテキストエディットに textHdl と length で指定したテキストを編集用にセットする。再表示は行わない。
マルチフォントテキストエディットの場合、カレントのスタイル情報で入力される。
疑似ハンドルは不可。
再配置が発生する。

Cの関数► int *TMSetTextH*(TEdit **tEHdl, char **textHdl, long length);
返り値はリザルトコード。

$A366　TMOpen

引　数►

long	textPtr	; 編集テキストの初期文字列へのポインタ (ASCIIZ 型)
long	maxLength	; 編集テキストの最大サイズ
long	dvRectPtr	; レクタングルレコードへのポインタ
word	mode	; 行揃えモード

0	左寄せ
1	センタリング
-1	右寄せ

word	lineHeight	; 改行幅のドット数

返り値► D0.L　リザルトコード
A0.L　テキストエディットレコードへのハンドル

機　能► textPtr で指定した文字列をあらかじめセットした maxLength までの文字列が編集可能なテキストエディットレコードを作成する。ディスティネーションレクタングル、ビューレクタングルは、どちらも dvRectPtr で指定したレクタングルとなる。

あらかじめ、目的のグラフレコードをカレントにしておく必要がある。

textPtr, dvRectPtr は疑似ポインタも可。

再配置が発生する。

新しく作成されるテキストエディットレコードの内容は次のとおり。

項目名	設定内容
ディスティネーションレクタングル	dvRectPtr で示されたレクタングル
ビューレクタングル	dvRectPtr で示されたレクタングル
編集テキストへのハンドル	テキストマンが作成したブロックのハンドル
編集テキストの最大サイズ	maxLength で指定したサイズ
編集テキストのブロックサイズ	テキストマンが作成したブロックのサイズ
改行幅	lineHeight で指定されたドット数
タブサイズ	48 ドット
行揃えモード	左揃え
グラフレコードへのポインタ	カレントグラフレコードへのポインタ
コントロールキーテーブルへのポインタ	デフォルト
カーソル表示ルーチンへのポインタ	デフォルト
クリップループルーチンへのポインタ	デフォルト
行頭テーブル	64 行分以上
その他	0

Cの関数► int *TMOpen*(const char *textPtr, long maxLength, Rect *dvRectPtr, int mode, int lineHeight, TEdit ***tEHdl);

結果は TEdit 型のハンドル tEHdl に格納される。

返り値はリザルトコード。

$A401　TMNew2

引　数►

```
long    destRectPtr    ; レクタングルレコードへのポインタ
long    viewRectPtr    ; ビューレクタングルへのポインタ
long    graphPtr       ; グラフレコードへのポインタ
```

返り値►

```
D0.L    リザルトコード
A0.L    テキストエディットレコードへのハンドル
```

機　能► destRectPtr, viewRectPtr で、それぞれ指定されたディスティネーショ

ンレクタングル、ビューレクタングルを持つテキストエディットレコードを作成する。
再配置が発生する。
新しく作成されるテキストエディットレコードの内容は次のとおり。

項目名	設定内容
ディスティネーションレクタングル	destRectPtr で示されたレクタングル
ビューレクタングル	viewRectPtr で示されたレクタングル
編集テキストへのハンドル	テキストマンが作成したブロックのハンドル
編集テキストの最大サイズ	0
編集テキストのブロックサイズ	0
改行幅	カレントグラフレコードにセットされているフォントの y 方向のサイズ
タブサイズ	48 ドット
行揃えモード	左揃え
グラフレコードへのポインタ	graphPtr で指定したグラフレコードへのポインタ
コントロールキーテーブルへのポインタ	デフォルト
カーソル表示ルーチンへのポインタ	デフォルト
クリップループルーチンへのポインタ	デフォルト
行頭テーブル	64 行分以上
その他	0

Cの関数► int *TMNew2*(Rect *destRectPtr, Rect *viewRectPtr, Graph *graphPtr, TEdit ***tEHdl);
結果は、TEdit 型のハンドル tEHdl に格納される。
返り値はリザルトコード。

$A460　　TMNextCodeIn

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
word　code　; キーコード
= 0　すべてのコード

返り値► D0.L　上位ワード ： キーコード、下位ワード ： ASCII コード
= 0　指定したコードのキーイベントは発見できなかった

機　能► 次のキーダウンイベントを先読みし、tEHdl で指定したテキストエディットレコードのキーアサインテーブルによってキーコードを変換したあと、code で指定したキーコードであるかどうかをチェックする。指定したコードを含むイベントが発生するまで、すべてのキーダウンイベントを取り除く。

Cの関数► int *TMNextCodeIn*(TEdit **tEHdl, int code);

返り値は、上位ワードがキーコード、下位ワードが ASCII コード。
イベントが発見できなかった場合は 0 。

$A462　TMSelReverse

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　selStart　; 開始位置（バイト位置）
long　selEnd　; 終了位置（バイト位置）
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットの selStart, selEnd で指定した範囲を反転表示する。ハイライト表示属性が負の場合は何もしない。
再配置が発生する。
Cの関数► int *TMSelReverse*(TEdit **tEHdl, long selStart, long selEnd);
返り値はリザルトコード。

$A463　TMTini

引　数► なし
返り値► D0.L　リザルトコード
機　能► テキストマンの終了処理を行う。
再配置が発生する。
Cの関数► int *TMTini*(void);
返り値はリザルトコード。

$A464　TMSetSelCal

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　selStart　; セレクト範囲の先頭位置のオフセット
long　selEnd　; セレクト範囲の終端位置のオフセット
long　selOffset　; 前のセレクト位置のオフセット
返り値► D0.L　リザルトコード
機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域を selStart, selEnd で指定した領域に新たに設定し、全体を計算し直す。再表示は行わない。selOffset は、前のセレクト位置の開始位置あるいは終了位置を指定する。開始位置を変更する場合は前の開始位置を、終了位置を変更する場合は前の終了位置を指定する。
再配置が発生する。
Cの関数► int *TMSetSelCal*(TEdit **tEHdl,long selStart,long selEnd,

long selOffset);
返り値はリザルトコード。

$A465　TMActivate2

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットレコードのハイライト表示レベルを＋1して、カーソルを表示するか、あるいはセレクト領域を反転表示する。
再配置が発生する。
Cの関数► int *TMActivate2*(TEdit **tEHdl);
返り値はリザルトコード。

$A466　TMDeactivate2

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
返り値► D0.L　リザルトコード
機　能► tEHdl で指定したテキストエディットレコードについて、カーソルを消去あるいはセレクト領域の反転表示を消したあと、ハイライト表示レベルを −1 する。
再配置が発生する。
Cの関数► int *TMDeactivate2*(TEdit **tEHdl);
返り値はリザルトコード。

$A467　TMCheckSel

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
long　selStart　；セレクト範囲の先頭位置のオフセット
long　selEnd　；セレクト範囲の終端位置のオフセット
long　selOffset　；前のセレクト位置のオフセット
返り値► D0.L　リザルトコード
機　能► tEHdl で指定されたテキストエディットレコードの編集テキストのセレクト領域を selStart, selEnd で指定した領域に新たに設定し、再表示を行う。selOffset は、前のセレクト位置の開始位置あるいは終了位置を指定する。開始位置を変更する場合は前の開始位置を、終了位置を変更する場合は前の終了位置を指定する。カーソル位置の再計算は行わない。
再配置が発生する。
Cの関数► int *TMCheckSel*(TEdit **tEHdl,long selStart,long selEnd,
long selOffset);

返り値はリザルトコード。

$A468　TMCalPoint2

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　offset　; バイト位置
long　buffPtr　; 座標情報が返るバッファ(12バイト)へのポインタ
返り値► A0.L　座標情報が返るバッファのアドレス
機　能► tEHdl で指定したテキストエディットの offset で指定したバイト位置の座標を計算し、buffPtr に返す。
バッファに返される情報の形式は以下のとおり。
+$00.L　水平座標
+$04.L　垂直座標
+$08.L　行揃えのための補正値
再配置が発生する。
Cの関数► void *TMCalPoint2*(TEdit **tEHdl, long offset, long *buffPtr);
返り値はない。

$A46A　TMISZen

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　offset　; バイト位置
返り値► D0.L　リザルトコード

0	シフトJISコードの1バイト目
1	シフトJISコードの2バイト目

A0.L　指定された位置のテキストのアドレス
機　能► tEHdl で指定した、テキストエディットの offset で指定したバイト位置がシフトJISコードの何バイト目かを調べる。
再配置が発生する。
Cの関数► int *TMISZen*(TEdit **tEHdl, long offset);
返り値は、調べた結果またはリザルトコード。

$A46B　TMSetDestOffset

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　offsetx　; 水平座標オフセット
long　offsety　; 垂直座標オフセット

返り値► D0.L　　リザルトコード

機　能► tEHdl で指定したテキストエディットの、ビューレクタングルに対するディスティネーションレクタングルのオフセットとして offsetx, offsety をセットし、再表示する。
再配置が発生する。

Cの関数► int *TMSetDestOffset*(TEdit **tEHdl, int offsetx, int offsety);
返り値はリザルトコード。

$A46C　　TMGetDestOffset

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル
long　buffPtr　　; 結果が返るバッファ(8バイト)へのポインタ

返り値► A0.L　　結果が返るバッファへのポインタ

機　能► tEHdl で指定した、テキストエディットのビューレクタングルに対するディスティネーションレクタングルのオフセットを、 buffPtr で指定したバッファに返す。
バッファに返される情報の形式は以下のとおり。
+$00.L　水平座標オフセット
+$04.L　垂直座標オフセット

Cの関数► void *TMGetDestOffset*(TEdit **tEHdl, long *buffPtr);
返り値はない。

$A46D　　TMGetSelect

引　数► long　tEHdl　　; テキストエディットレコードへのハンドル
long　buffPtr　　; 結果が返るバッファ(12 バイト) へのポインタ

返り値► D0.L　　リザルトコード
A0.L　　結果が返るバッファへのポインタ

機　能► tEHdl で指定したテキストエディットの現在のセレクト状態を buffPtr で指定したバッファに返す。
バッファに返される情報の形式は以下のとおり。
+$00.L　セレクト開始位置
+$04.L　セレクト終了位置
+$08.L　現在のセレクト位置

Cの関数► int *TMGetSelect*(TEdit **tEHdl, long *buffPtr);
返り値はリザルトコード。

$A46E TMEventW 2.0

引　数► long tEHdl ；テキストエディットレコードへのハンドル
long eventRecPtr ；イベントレコードへのポインタ

返り値► D0.L リザルトコード

0	編集しなかった
1	編集した

機　能► tEHdl で指定したテキストエディットレコードに格納されているグラフレコードへのポインタ（teInPort）がウィンドウレコードへのポインタであると仮定し、アップデートリージョンの範囲内を再描画し、再描画した範囲はアップデートリージョンから取り除く。その後、$A31C TMEvent と同様な処理を行う。
アップデートリージョンを操作するので、$A20D WMUpdate 後には使用することができない。
再配置が発生する。

Cの関数► int *TMEventW*(TEdit **tEHdl, TsEvent *eventRecPtr);
返り値は、編集結果またはリザルトコード。

$A46F TMUpDateExist 2.0

引　数► long tEHdl ；テキストエディットレコードへのハンドル
long mode ＝0 描画した範囲をアップデートリージョンから取り除かない
≠0 描画した範囲をアップデートリージョンから取り除く

返り値► D0.L リザルトコード

0	アップデートリージョンが存在しない
1	アップデートリージョンを描画した

機　能► tEHdl で指定したテキストエディットレコードに格納されているグラフレコードへのポインタ（teInPort）がウィンドウレコードへのポインタであると仮定し、アップデートリージョンの範囲内を再描画し、mode が 0 以外の場合は再描画した範囲をアップデートリージョンから取り除く。
アップデートリージョンを操作するので、$A20D WMUpdate 後には使用できない。
再配置が発生する。

Cの関数► int *TMUpDateExist*(TEdit **tEHdl, BOOLEAN mode);
返り値は、アップデート結果またはリザルトコード。

$A470　TMNewM　3.0

引　数► long　destRectPtr　; ディスティネーションレクタングルへのポインタ
long　viewRectPtr　; ビューレクタングルへのポインタ
long　graphPtr　; グラフレコードへのポインタ
long　styleRecPtr　; スタイル情報レコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　マルチフォントテキストエディットレコードへのハンドル

機　能► destRectPtr, viewRectPtr でそれぞれ指定されたディスティネーションレクタングル、ビューレクタングルを持つマルチフォントテキストエディットレコードを作成する。
styleRecPtr には、カレントのフォント情報となるスタイル情報レコードを指定する。
再配置が発生する。

Cの関数► int *TMNewM*(Rect *destRectPtr, Rect *viewRectPtr, Graph *graphPtr, TEStyle *styleRecPtr, MTEdit ***mTEHdl);
結果は、 MTEdit 型のハンドル mTEHdl に格納される。
返り値はリザルトコード。

$A471　TMSetTextM　3.0

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　textHdl　; 文字列へのハンドル
long　length　; 文字列のバイト数
long　styleCellHdl　; スタイル情報のセルへのハンドル
byte　mode1　= 0　textHdl をそのまま編集テキストにする
≠ 0　textHdl を別のブロックに複写後、編集テキストにする
byte　mode2　= 0　styleCellHdl をそのままスタイル情報にする
≠ 0　styleCellHdl を別のブロックに複写後、スタイル情報にする

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したテキストエディットに textHdl と length で指定したテキストを編集用にセットし、styleCellHdl で指定したスタイル情報を付加する。再表示は行わない。
mode1,mode2 に 0 以外を指定した場合、それぞれ textHdl,styleCellHdl

は疑似ハンドルも可。
再配置が発生する。

Cの関数► int *TMSetTextM*(MTEdit **mTEHdl, char **textHdl, long length, void **styleCellHdl, int mode1, int mode2);
返り値はリザルトコード。

$A472 TMSetDefKind 3.0

引　数► long mTEHdl ；マルチフォントテキストエディットレコードへのハンドル
long styleRecPtr ；スタイル情報レコードへのポインタ

返り値► D0.L リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットレコードに、styleRecPtr で指定したスタイル情報をカレントとしてセットする。
最終行の改行幅に変更があった場合（最終行が空行の場合）は“[EOF]”を再表示する。
再配置が発生する。

Cの関数► int *TMSetDefKind*(MTEdit **mTEHdl, TEStyle *styleRecPtr);
返り値はリザルトコード。

$A473 TMGetStyle 3.0

引　数► long mTEHdl ；マルチフォントテキストエディットレコードへのハンドル
long lStyleRecPtr ；文字サイズ付きスタイル情報レコードへのポインタ
long offset ；バイト位置

返り値► D0.L リザルトコード
A0.L スタイル情報セル中の該当レコードへのポインタ

機　能► mTEHdl で指定したマルチフォントテキストエディット中の、offset で指定したバイト位置のスタイル情報を lStyleRecPtr で指定したスタイル情報レコードに格納する。オプション情報は無視される。
A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が発生すると無効になる。

Cの関数► int *TMGetStyle*(MTEdit **mTEHdl, TELStyle *lStyleRecPtr, long offset);
返り値はリザルトコード。

$A475 TMGetStyles 3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　styleCellHdl　；スタイル情報セルが返る再配置可能ブロックへのハンドル
long　selStart　；開始位置
long　selEnd　；終了位置

返り値► D0.L　スタイル情報セルのバイト数 −8 /リザルトコード
A0.L　スタイル情報セルへのハンドル

機　能► mTEHdl で指定したマルチフォントテキストエディットの selStart〜selEnd 間のスタイル情報を、styleCellHdl で指定したブロックに格納する（ブロックのサイズは調整される）。
styleCellHdl に 0 を指定した場合、テキストマンがヒープ中に作成する。
再配置が発生する。

Cの関数► void ***TMGetStyles*(MTEdit **mTEHdl, void **styleCellHdl, long selStart, long selEnd);
返り値はスタイル情報セルへのハンドル。
エラーが発生した場合、NULL が返る。

$A476 TMChangeStyle 3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　styleRecPtr　；スタイル情報レコードへのポインタ
word　flags　；設定する情報を指定するフラグ

bit0	フォントカインドを設定する
bit1	フォントフェイスを設定する
bit2	x 方向のフォントサイズを設定する
bit3	y 方向のフォントサイズを設定する

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットのセレクト範囲に styleRecPtr で指定したスタイル情報を設定し、再表示する。このとき、flags でビットを 1 にした情報についてのみ設定が行われる。
再配置が発生する。

Cの関数► int *TMChangeStyle*(MTEdit **mTEHdl, TEStyle *styleRecPtr, int flags);
返り値はリザルトコード。

$A477　TMChangeFace　3.0

引　数► long　mTEHdl　;マルチフォントテキストエディットレコードへのハンドル
word　face　;フォントフェイス
word　mask　;マスク

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットのセレクト範囲に face で指定されたフォントフェイスを設定し、再表示する。このとき、mask でビットを 1 にしたフォントフェイス情報についてのみ、設定が行われる。
再配置が発生する。

Cの関数► int *TMChangeFace*(MTEdit **mTEHdl, int face, int mask);
返り値はリザルトコード。

$A478　TMSetColor　3.0

引　数► long　mTEHdl　;マルチフォントテキストエディットレコードへのハンドル
word　foreColor　;フォアグラウンドカラー
word　backColor　;バックグラウンドカラー
word　aPage　;アクセスページ

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエデットの、デフォルトのフォアグラウンドカラー、バックグラウンドカラー、アクセスページをそれぞれ foreColor, backColor, aPage にする。
再表示はしない。

Cの関数► int *TMSetColor*(MTEdit **mTEHdl, int foreColor, int backColor, int aPage);
返り値はリザルトコード。

$A479　TMSetMode　3.0

引　数► long　mTEHdl　;マルチフォントテキストエディットレコードへのハンドル
word　mode　;描画モード
word　hColor　;反転文字色
word　exMode

bit0	ワードブレーク処理
bit1	タブ表示
bit8	通常は 1

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットレコードの拡張編集モードを設定する。設定後、$A464 TMSetSelCal で再計算を行い、$A313 TMUpDate で再描画する必要がある。

Cの関数► int *TMSetMode*(MTEdit **mTEHdl,int mode,int hColor, int exMode);
返り値はリザルトコード。

$A47A　TMPutScrapM　3.0

引　数► long　length　; 文字列のバイト数
long　strHdl　; 文字列へのハンドル
long　styleCellHdl　; スタイル情報セルへのハンドル

返り値► D0.L　リザルトコード

機　能► strHdl で指定した length バイトの文字列、styleCellHdl で指定したスタイル情報セルを、テキストマンスクラップに格納する。
両方のハンドルに疑似ハンドルを使用することが可能。
再配置が発生する。

Cの関数► int *TMPutScrapM*(long length, char **strHdl, void **styleCellHdl);
返り値はリザルトコード。

$A47B　TMInsertM　3.0

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　strPtr　; 文字列へのポインタ
long　length　; 文字列のバイト数
long　styleCellHdl　; スタイル情報セルへのハンドル

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットのセレクト領域と strPtr で指定した文字列を置き換えて再表示する。その際、styleCellHdl で指定したスタイル情報が同時に設定される。
再配置が発生する。

Cの関数► int *TMInsertM*(MTEdit **mTEHdl,char *strPtr,long length, void **styleCellHdl);
返り値はリザルトコード。

$A47C　TMStrM　3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　strPtr　；文字列へのポインタ
long　length　；文字列のバイト数
long　hisRecPtr　；編集履歴レコードへのポインタ
long　styleCellHdl　；スタイル情報セルへのポインタ

返り値► D0.L　リザルトコード

0	編集しなかった
1	編集した

機　能► mTEHdl で指定されたマルチフォントテキストエディットレコードの編集テキストのセレクト領域を、strPtr で指定した文字列と置き換え、編集履歴レコードを作成する。カーソル位置の再計算は行われない。その際、styleCellHdl で指定したスタイル情報が同時に設定される。CR($0D) 以外の制御コードを含めた場合、文字として扱われる。再表示は行わない。
再配置が発生する。

Cの関数► int *TMStrM*(MTEdit **mTEHdl, char *strPtr, long length, TEHis *hisRecPtr, void **styleCellHdl);
返り値は、編集結果またはリザルトコード。

$A47D　TMSetStyles　3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　styleCellHdl　；スタイル情報セルへのポインタ

返り値► D0.L　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットのセレクト領域に、styleCellHdl で指定したスタイル情報を設定し、再表示する。
再配置が発生する。

Cの関数► int *TMSetStyles*(MTEdit **mTEHdl, void **styleCellHdl);
返り値はリザルトコード。

$A47E　TMGetExStyles　3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　lStyleRecPtr　；文字サイズ付きスタイル情報レコードへのポインタ
long　offset　；バイト位置

long　optRecPtr　;オプション情報レコードへのポインタ

返り値► D0.L　リザルトコード

A0.L　スタイル情報セル中の該当レコードへのポインタ

機　能► mTEHdl で指定したマルチフォントテキストエディットレコードの offset で指定したバイト位置のスタイル情報を返す。lStyleRecPtr で指定したスタイル情報レコードにはスタイル情報が、optRecPtr で指定したオプション情報レコードにはオプション情報が格納される。

offset に編集テキスト以外の位置を指定すると、カレントのスタイル情報を返す。

A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が発生すると無効になる。

Cの関数► int *TMGetExStyles*(MTEdit **mTEHdl,TELStyle *lStyleRecPtr, long offset, TEOption *optRecPtr);

返り値はリザルトコード。

$A47F　TMGetScrap　3.0

引　数► なし

返り値► D0.L　文字列のバイト数

A0.L　テキストマンスクラップへのポインタ

機　能► テキストマンスクラップへのポインタを返す。テキストマンスクラップの構造は以下のとおり。

+$00.L　データのバイト数

+$04.L　データへのハンドル

+$08.W　格納したタスクのID

+$0A.L　文字情報へのハンドル

Cの関数► TEScrap **TMGetScrap*(void);

返り値はテキストマンスクラップへのポインタ。

$A480　TMGetLineWidth　3.0

引　数► long　tEHdl　;テキストエディットレコードへのハンドル

long　line　;行位置

返り値► D0.L　行の幅（ドット数）/リザルトコード

機　能► tEHdl で指定したテキストエディットの、line で指定した行の幅（ドット数）を返す。

Cの関数► int *TMGetLineWidth*(TEdit **tEHdl, long line);

返り値は、行の幅またはリザルトコード。

$A481　TMGetLineHeight　3.0

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　line　; 行位置
返り値► D0.L　改行幅（ドット数）/リザルトコード
機　能► tEHdl で指定したテキストエディットの、 line で指定した行の改行幅（ドット数）を返す。
Cの関数► int *TMGetLineHeight*(TEdit **tEHdl, long line);
返り値は、改行幅またはリザルトコード。

$A482　TMLineToHeight　3.0

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　line　; 行位置
返り値► D0.L　垂直座標
機　能► tEHdl で指定したテキストエディットの、 line で指定した行の垂直座標（ロングワード）を返す。
Cの関数► int *TMLineToHeight*(TEdit **tEHdl, long line);
返り値は、垂直座標。

$A483　TMAdjustHeight　3.0

引　数► long　tEHdl　; テキストエディットレコードへのハンドル
long　y　; 垂直座標
返り値► D0.L　行位置
A0.L　補正後の垂直座標
機　能► tEHdl で指定したテキストエディットの、y で指定した垂直座標にあたる行位置と補正した垂直座標を返す。
Cの関数► int *TMAdjustHeight*(TEdit **tEHdl, long y);
返り値は行位置。

$A484　TMChangeExStyle　3.0

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　styleRecPtr　; スタイル情報レコードへのポインタ
word　flags　; 設定する情報を指定するフラグ

bit0	フォントカインドを設定する
bit1	フォントフェイスを設定する
bit2	x 方向のフォントサイズを設定する
bit3	y 方向のフォントサイズを設定する

long　　mask　　　　　　; オプション情報のマスク

bit0	行揃えを設定する
bit1	文字色を設定する
bit2	描画モードを設定する
bit3	文字の高さを設定する
bit4	文字間ピッチを設定する
bit5	改行幅を指定する
bit6〜7	システム予約
bit8	セルを文字のかわりに描画
bit9	セルを文字の大きさに描画（網掛け）
bit10	セルを文字の下に描画（下線）
bit11	セルを文字の上に描画（ルビ）
bit12〜15	システム予約
bit16〜23	システム予約
bit24	フォントカインド変更禁止設定
bit25	フォントフェイス変更禁止設定
bit26	x 方向のフォントサイズ変更禁止設定
bit27	y 方向のフォントサイズ変更禁止設定
bit28〜29	システム予約
bit30	ブロック単位で表示
bit31	フォント情報の連結禁止設定

返り値► D0.L　　リザルトコード

機　能► mTEHdl で指定したマルチフォントテキストエディットのセレクト領域に styleRecPtr で指定したスタイル情報を設定し、再表示する。このとき、flags でビットを 1 にした情報についてのみ設定が行われる。オプション情報は mask で指定されている項目のみ設定が行われる。
再配置が発生する。

Cの関数► int *TMChangeExStyle*(MTEdit **mTEHdl,TEStyle *styleRecPtr, int flags, long mask);
返り値はリザルトコード。

$A485　　TMAnalyzeExStyle　　3.0

引　数► long　　optRecPtr　　　; オプション情報レコードへのポインタ
long　　lStyleRecPtr　　; 文字サイズ付きスタイル情報レコードへのポインタ

返り値► D0.L　　リザルトコード

機　能► lStyleRecPtr で指定したスタイル情報のオプション情報を、optRecPtr

で指定したオプション情報レコードに格納する。

Cの関数► int *TMAnalyzeExStyle*(TEOption *optRecPtr, TELStyle *lStyleRecPtr);
返り値はリザルトコード。

$A486 TMSetEditMode 3.0

引　数► long tEHdl ；テキストエディットレコードへのハンドル
word flags ；編集モード

bit0	改行コード表示
bit1	[EOF] 表示
bit2	コントロールコードを組文字表示
bit3	未使用
bit4	編集禁止
bit5	下揃え
bit6	LF で改行
bit7	CR で改行

返り値► D0.L リザルトコード

機　能► tEHdl で指定したテキストエディットの編集モードを設定する。設定後、$A464 TMSetSelCal で再計算を行い、$A313 TMUpDate で再描画する必要がある。

Cの関数► int *TMSetEditMode*(TEdit **tEHdl, int flags);
返り値はリザルトコード。

$A487 TMCellToFont 3.0

引　数► long cellHdl ；セルレコードへのハンドル
long styleCellHdl ；スタイル情報セルへのハンドル
long flags ；オプション情報

bit0	行揃えを設定する
bit1	文字色を設定する
bit2	描画モードを設定する
bit3	文字の高さを設定する
bit4	文字間ピッチを設定する
bit5	改行幅を指定する
bit6〜7	システム予約
bit8	セルを文字のかわりに描画
bit9	セルを文字の大きさに描画（網掛け）
bit10	セルを文字の下に描画（下線）
bit11	セルを文字の上に描画（ルビ）
bit12〜15	システム予約
bit16〜23	システム予約
bit24	フォントカインド変更禁止設定
bit25	フォントフェイス変更禁止設定
bit26	x 方向のフォントサイズ変更禁止設定
bit27	y 方向のフォントサイズ変更禁止設定
bit28〜29	システム予約
bit30	ブロック単位で表示
bit31	フォント情報の連結禁止設定

long　sizePt　; フォントサイズを意味するポイント

返り値► D0.L　リザルトコード

A0.L　スタイル情報セルへのハンドル

機　能► cellHdl で指定したセルレコードの内容を styleCellHdl で指定したスタイル情報セルに格納する。このとき、flags でビットを 1 にした情報についてのみ設定が行われる。セルのなかのイメージやスクリプトは sizePt で指定したサイズになる。スタイル情報セルとなる再配置可能ブロックのサイズは伸縮される。styleCellHdl として 0 を指定すると、テキストマンがヒープ上に作成する。

cellHdl は疑似ハンドルも可。

再配置が発生する。

Cの関数► void ***TMCellToFont*(void **cellHdl, void **styleCellHdl, long flags, LPoint sizePt);

返り値はスタイル情報セルへのハンドル。

エラーが発生した場合、NULL が返る。

$A488　TMScaleSet　3.0

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル

word　scaleX　;x 方向の倍率（固定小数点数）

	word	scaleY	;y 方向の倍率（固定小数点数）
返り値►	D0.L	リザルトコード	

機　能► mTEHdl で指定したマルチフォントテキストエディットの表示倍率を scaleX, scaleY で指定する。ビューレクタングル以外の座標データが拡大/縮小される。

scaleX, scaleY で使用する固定小数点数は、整数部 6 ビット、小数部 10 ビット（つまり、倍率を 1024 倍した数を指定する）。

再配置が発生する。

Cの関数► int *TMScaleSet*(MTEdit **mTEHdl, LPoint scale);

scale は、 scaleX と scaleY を組み合わせた LPoint。

返り値はリザルトコード。

$A489　TMScaleStyles　3.0

引　数►	long	mTEHdl	; マルチフォントテキストエディットレコードへのハンドル
	long	sStyleCellHdl	; スタイル情報セルへのハンドル
	word	scaleX	;x 方向の倍率（固定小数点数）
	word	scaleY	;y 方向の倍率（固定小数点数）
	long	dStyleCellHdl	; 結果が返るスタイル情報セルへのハンドル
返り値►	D0.L	リザルトコード	
	A0.L	スタイル情報セルへのハンドル	

機　能► sStyleCellHdl で指定したスタイル情報セルの内容を、scaleX, scaleY で指定した倍率に伸縮し、その結果を dStyleCellHdl で指定されたメモリブロックに格納する。メモリブロックのサイズは伸縮される。dStyleCellHdl として 0 を指定すると、テキストマンがヒープ上に作成する。

sStyleCellHdl と dStyleCellHdl には同一のハンドルを指定できる。

sStyleCellHdl は疑似ハンドルも可。

scaleX, scaleY で使用する固定小数点数は、整数部 6 ビット、小数部 10 ビット（つまり、倍率を 1024 倍した数を指定する）。

再配置が発生する。

Cの関数► void ***TMScaleStyles*(MTEdit **mTEHdl, void **sStyleCellHdl, LPoint scale, void **dStyleCellHdl);

scale は、scaleX と scaleY を組み合わせた LPoint。

返り値はスタイル情報セルへのハンドル。

エラーが発生した場合、NULL が返る。

$A48A　TMBundleExStyle　3.0

引　数►	long	optRecPtr	; オプション情報レコードへのポインタ

long　lStyleRecPtr　；文字サイズ付きスタイル情報レコードへのポインタ

返り値► D0.L　リザルトコード

機　能► optRecPtrで指定したオプション情報をスタイル情報に変換し、lStyleRecPtrで指定したバッファに格納する。バッファには十分な領域を用意する必要がある。

Cの関数► int *TMBundleExStyle*(TEOption *optRecPtr, TELStyle *lStyleRecPtr);
返り値はリザルトコード。

$A48B　TMSetLineHeight　3.0

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
word　height　；改行幅

返り値► D0.L　リザルトコード

機　能► tEHdl で指定したテキストエディット全体の改行幅を height に設定する。設定後、$A464 TMSetSelCalで再計算を行い、$A313 TMUpDateで再描画する必要がある。

Cの関数► int *TMSetLineHeight*(TEdit **tEHdl, int height);
返り値はリザルトコード。

$A48C　TMSetTabSize　3.0

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
word　tabSize　；タブのドット数

返り値► D0.L　リザルトコード

機　能► tEHdl で指定したテキストエディットのタブの幅を tabSize に設定する。設定後、$A464 TMSetSelCalで再計算を行い、$A313 TMUpDateで再描画する必要がある。

Cの関数► int *TMSetTabSize*(TEdit **tEHdl, int tabSize);
返り値はリザルトコード。

$A48D　TMGetStr　3.0

引　数► long　tEHdl　；テキストエディットレコードへのハンドル
long　buffPtr　；バッファへのポインタ
long　startOffset　；開始位置
long　endOffset　；終了位置

返り値► D0.L　文字列のバイト数/リザルトコード
A0.L　文字列の終端のアドレス ＋1

機　能► tEHdl で指定したテキストエディットの、startOffset ～ endOffset の編集テキストを buffPtr に格納する。テキストがキャッシュ中にある場合、直接キャッシュから転送するので、フラッシュする必要はない。

Cの関数► int *TMGetStr*(TEdit **tEHdl, char *buffPtr, long startOffset, long endOffset);
返り値は、文字列のバイト数またはリザルトコード。

$A48E　TMScalePtSet　3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　pt　；ポイント

返り値► D0.L　変換後のポイント

機　能► pt で指定したポイントを、mTEHdl で指定したマルチフォントテキストエディットの表示倍率にあわせて変換する。

Cの関数► LPoint *TMScalePtSet*(MTEdit **mTEHdl, LPoint pt);
返り値は変換後のポイント。

$A48F　TMScalePtReSet　3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　pt　；ポイント

返り値► D0.L　ポイント

機　能► $A48E TMScalePtSet によって変換したポイントを元に戻す。

Cの関数► LPoint *TMScalePtReSet*(MTEdit **mTEHdl, LPoint pt);
返り値はポイント。

$A490　TMGetDefKind　3.0

引　数► long　mTEHdl　；マルチフォントテキストエディットレコードへのハンドル
long　lStyleRecPtr　；文字サイズ付きスタイル情報レコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　スタイル情報セル中の該当レコードへのポインタ

機　能► mTEHdl で指定したマルチフォントテキストエディットのカレントのスタイル情報を lStyleRecPtr で指定したバッファに格納する。オプション情報は無視される。
A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が

発生すると無効になる。
再配置が発生する。

Cの関数► int *TMGetDefKind*(MTEdit **mTEHdl,
TELStyle *lStyleRecPtr);
返り値はリザルトコード。

$A494 TMVer 3.1

引　数► なし
返り値► D0.L テキストマンのバージョン
機　能► テキストマンのバージョンを返す。SX-WINDOW ver.3.1 のテキストマンでは、返り値は 0x0302 となる。
SX-WINDOW ver.3.01 以前では使用することができない。
Cの関数► int *TMVer*(void);
返り値はテキストマンのバージョン。

$A495 TMSetPage 3.1

引　数► long mTEHdl ;マルチフォントテキストエディットレコードへのハンドル
word page ;1ページのサイズ
返り値► D0.L リザルトコード
機　能► mTEHdl で指定したテキストエディットの1ページのサイズを page で指定したドット数にする。
再配置が発生する。
Cの関数► int *TMSetPage*(MTEdit **mTEHdl, short page);
返り値はリザルトコード。

$A496 TMHeightToPage 3.1

引　数► long mTEHdl ;マルチフォントテキストエディットレコードへのハンドル
long height ;垂直座標
返り値► D0.L ページ位置/リザルトコード
A0.L ページ情報へのポインタ
機　能► mTEHdl で指定したテキストエディットについて、垂直座標 height で指定した位置に関するページ情報を返す。
A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が発生すると無効になる。

ページ情報の構造は以下のとおり。

```
typedef struct {
        long        height;     /* 垂直座標 */
        long        line;       /* 行位置 */
        long        offset;     /* バイト位置 */
        long        oheight;    /* 垂直座標(100%表示時) */
} TEPage;
```

Cの関数► int *TMHeightToPage*(MTEdit **mTEHdl, int height, TEPage **pinfo);
結果は、TEPage 型のポインタ pinfo に格納される。
返り値はページ位置またはリザルトコード。

$A497 TMOffsetToPage 3.1

引　数► long mTEHdl ;マルチフォントテキストエディットレコードへのハンドル
long offset ;バイト位置

返り値► D0.L ページ位置/リザルトコード
A0.L ページ情報へのポインタ

機　能► mTEHdl で指定したテキストエディットについて、offset で指定したバイト位置に関するページ情報を返す。
A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が発生すると無効になる。

Cの関数► int *TMOffsetToPage*(MTEdit **mTEHdl, int offset, TEPage **pinfo);
結果は、TEPage 型のポインタ pinfo に格納される。
返り値はページ位置またはリザルトコード。

$A498 TMPageToLine 3.1

引　数► long mTEHdl ;マルチフォントテキストエディットレコードへのハンドル
long page ;ページ位置

返り値► D0.L 行位置/リザルトコード
A0.L ページ情報へのポインタ

機　能► mTEHdl で指定したテキストエディットについて、page で指定したページ位置の行位置とページ情報を返す。
A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が発生すると無効になる。

Cの関数► int *TMPageToLine*(MTEdit **mTEHdl, int page,

TEPage **pinfo);
結果は、TEPage 型のポインタ pinfo に格納される。
返り値は行位置またはリザルトコード。

$A499　TMLineToPage　3.1

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　line　; 行位置
返り値► D0.L　ページ位置/リザルトコード
A0.L　ページ情報へのポインタ
機　能► mTEHdl で指定したテキストエディットについて、line で指定した行位置のページ位置とページ情報を返す。
A0.L に返るポインタは再配置可能ブロック中へのポインタなので、再配置が発生すると無効になる。
Cの関数► int *TMLineToPage*(MTEdit **mTEHdl, int line,
TEPage **pinfo);
結果は、TEPage 型のポインタ pinfo に格納される。
返り値はページ位置またはリザルトコード。

$A49A　TMTextWidth3　3.1

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　strPtr　; 文字列へのポインタ
long　offset　; 文字列のオフセット
word　length　; 文字列のバイト数
word　startPoint　; 文字列のローカル座標—ディステネーションレクタングルの水平方向オフセット
返り値► D0.L　文字列の占める幅（ドット数）/リザルトコード
機　能► mTEHdl で指定したマルチフォントテキストエディットレコードの環境下で、startPoint で指定した位置から strPtr, offset, length で指定される文字列を描画する場合、それが占める幅（ドット数）を計算し、結果を返す。
拡大/縮小表示をしている場合でも 100% 時の幅を返す。
再配置が発生する。
Cの関数► int *TMTextWidth3*(MTEdit **mTEHdl, const char *strPtr,
long offet, int length, int startPoint);
返り値は、文字列の占める幅またはリザルトコード。

$A49B　TMLineToRHeight　3.1

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　line　; 行位置
返り値► D0.L　垂直座標（ロングワード）
機　能► mTEHdl で指定したマルチフォントテキストエディットについて、line で指定した行位置の垂直座標を返す。
拡大/縮小表示をしている場合でも 100% 時の幅を返す。
あらかじめ、$A495 TMSetPage を実行しておく必要がある。
再配置が発生する。
Cの関数► int *TMLineToRHeight*(MTEdit **mTEHdl, long line);
返り値は垂直座標。

$A49D　TMGetLineRHeight　3.1

引　数► long　mTEHdl　; マルチフォントテキストエディットレコードへのハンドル
long　line　; 行位置
返り値► D0.L　改行幅（ドット数)/リザルトコード
機　能► mTEHdl で指定したマルチフォントテキストエディットについて、line で指定した行位置の改行幅を返す。
拡大/縮小表示をしている場合でも 100% 時の幅を返す。
あらかじめ、$A495 TMSetPage を実行しておく必要がある。
Cの関数► int *TMGetLineRHeight*(MTEdit **mTEHdl, long line);
返り値は改行幅またはリザルトコード。

タスクマン

#include <TASK.H>

$A34C　TSInitTsk

引　数► long　memStart　；ヒープゾーン先頭アドレス
long　memEnd　；ヒープゾーン終了アドレス
long　pathPtr　；カレントパスを示す文字列（ASCIIZ 型）へのポインタ
byte　mode1　；シェルのリリース番号（1～）
byte　mode2　；システムリソースヒープゾーン作成フラグ

0	作成しない
≠0	作成する

word　ver　；シェルのバージョン

返り値► D0.L　ヒープゾーンへのポインタ/リザルトコード
A0.L　システムリソースマップへのハンドル

機　能► memStart から memEnd までのメモリの先頭の領域をタスクマン用ワークとして初期化し、残りをヒープゾーンとしてメモリマンで初期化する。mode2 として 0 以外を指定した場合は、システムリソース用のヒープゾーンを作成し、そこにシステムリソース（'SYSTEM.LB'）をすべて読み込んだあと、あらためてもう1つヒープゾーンを作成する。システムリソースは path で指定したパスから検索を開始する。path は 0 を指定することで省略できる。
ヒープゾーン作成後、画面モードを初期化して、すべてのマネージャを初期化する。

Cの関数► Heap *_TSInitTsk_(void *memStart, void *memEnd, const char *pathPtr, int mode1, int mode2, int ver, Handle *sysRes);
返り値はヒープゾーンへのポインタ。ハンドル sysRes にはシステムリソースマップへのハンドルが返る。

$A34E　TSInitCrtM

引　数► なし

返り値► D0.L　前の画面モード（IOCS の CTRMOD の値と同等）/リザルトコード

機　能► 画面モードを初期化する。SRAM の SX-WINDOW 用ワークが初期化されてい

ない場合、初期化する。
再配置が発生する。

Cの関数► int *TSInitCrtM*(void);
返り値は前の画面モード、またはリザルトコード。

$A34F TSTiniCrtM

引 数► word mode ; 画面モード (IOCS の CTRMOD の値と同等)

返り値► D0.L リザルトコード

機 能► 画面モードを mode で示される値に戻す。
再配置が発生する。

Cの関数► int *TSTiniCrtM*(int mode);
返り値はリザルトコード。

$A351 TSFock

引 数►

byte	mode1	; 起動モード1
byte	mode2	; 起動モード2
long	taskID/filename	; ファイル名またはタスクID
long	commandPtr	; コマンドライン文字列 (LASCII 型) へのポインタ
long	environPtr	; 環境へのポインタ
long	Type	; 起動するリソースのタイプ
word	ID	; 起動するリソースのID

返り値► D0.L タスクID/リザルトコード

機 能► タスクを起動する。
タスクには commandPtr で指定したコマンドライン文字列と、environPtr で指定した環境が渡される。environPtr として 0 を指定すると、タスクマンの環境を使用することになる。
mode1, mode2 の指定によって、実行ファイル、リソース、メモリの3種類からタスクを起動することができる。いずれもモジュールヘッダが必要であり、Human68k の実行ファイルなどはサポートされない。
再配置が発生する。

- mode2 = 0 の場合 : **ファイルからの起動**

  filename で指定したファイルから起動する。Type, ID は意味を持たない。

  mode1 = 0

  　ファイルをロード・実行する。

  mode1 = 1

ファイルをロード・実行する。実行終了後、コード領域を開放しない。

mode1 = 2

ファイルをロードする。実行はしない。

mode1 = 3

taskIDで指定したタスクIDで、mode1 = 2 でロードしたファイルを実行する。

● mode2 = 1 の場合 ： **リソースからの起動(ローカルリソース対応)**

Type, IDで指定したリソースを読み込んで起動する。読み込まれたリソースがXタイプの実行形式であった場合はリソースマンの管理から外される。

mode1 = 0

リソースをロード・実行する。

mode1 = 1

リソースをロード・実行する。実行終了後、コード領域を開放しない。

mode1 = 2

リソースをロードする。実行はしない。

mode1 = 3

taskIDで指定したタスクIDで、mode1 = 2でロードしたリソースを実行する。

● mode2 = 2の場合 ： **メモリからの起動**

taskIDで指定したタスク（リエントラントである必要がある）と同じものを、コード領域を共有して起動する。Type, IDは意味を持たない。

mode1 = 0

タスクをタスク管理テーブル上で複写して実行する。

mode1 = 2

タスクをタスク管理テーブル上で複写する。実行はしない。

mode1 = 3

taskIDで指定したタスクIDで、mode1 = 2で複写したタスクを実行する。

Cの関数► int *TSFock*(int mode1, int mode2, char *filename, const char *commandPtr, const char *environPtr, long Type, int ID);

返り値はタスクIDまたはリザルトコード。

$A352 TSExit

引　数► long code ；リザルトコード

返り値► なし（戻ってこない）

機　能► code で指定したリザルトコードを持って、タスクを終了する。
アプリケーションが作成したメモリブロックは廃棄されないので、必ず廃棄しておく必要がある。
再配置が発生する。
原則としてＣ言語で書かれたプログラムでは使用してはならない。

$A353　　TSFockB

引　数► byte　mode1　　; 起動モード 1
byte　mode2　　; 起動モード 2
long　filename　　; ファイル名
long　commandPtr　　; コマンドライン文字列（LASCII 型）へのポインタ
long　environPtr　　; 環境へのポインタ
long　dFilename　　; 実際に起動したファイル名が返るバッファ（90 バイト）のアドレス

返り値► D0.L　　タスク ID/リザルトコード

機　能► filename で指定したファイル、あるいはビルトインコマンドからタスクを起動する。すでにメモリに存在するリエントラントなタスクの場合、タスク管理テーブル上で複写して実行する。
タスクには commandPtr で指定したコマンドライン文字列と、environPtr で指定した環境が渡される。environPtr として 0 を指定すると、タスクマンの環境を使用することになる。
mode1 は $A351 TSFock と同様な意味を持つ。
mode2 として 0 以外を指定した場合、起動した（しようとした）ファイル名を dFilename に返す。
filename にはワイルドカードが使用可能。
再配置が発生する。

Cの関数► int *TSFockB*(int mode1, BOOLEAN mode2, const char *filename,const char *commandPtr,const char *environPtr, char *dFilename);
返り値は、タスク ID またはリザルトコード。キャラクタ型の配列 dFilename[90] には、起動したファイルの名前が返される。

$A355　　TSFockSItem

引　数► long　sItemHdl　　; セルリストへのハンドル

返り値► D0.L　　タスク ID/リザルトコード

機　能► sItemHdl で指定したセルリストのなかのアイコン管理レコードを参照して、そのすべてを起動する。返り値のタスク ID は、最後に起動したものが返る。

再配置が発生する。

Cの関数► int *TSFockSItem*(Handle sItemHdl);
返り値は、タスク ID またはリザルトコード。

$A356 TSFockIcon

引　数► long ISRecPtr ; アイコン管理レコードへのポインタ
byte mode1 ; 起動モード 1
byte mode2 ; 起動モード 2
long dFilename ; 実際に起動したファイル名が返るバッファ (90 バイト) のアドレス
long command ; コマンドライン文字列 (LASCII 型) へのポインタ

返り値► D0.L タスク ID/リザルトコード

機　能► ISRecPtr で指定したアイコン管理レコードを参照してタスクを起動する。
mode1 は$A351 TSFock と同様の意味を持つ。
mode2 として 0 以外を指定した場合、起動した (しようとした) ファイル名とコマンドラインを dFilename に返す。
起動時に [OPT.1] キーが押されている場合は、cshell ウィンドウを開いてファイル名とコマンドラインの編集を行う。
再配置が発生する。

Cの関数► int *TSFockIcon*(IcState *ISRecPtr, int mode1, BOOLEAN mode2, char *dFileName, char *command);
返り値は、タスク ID またはリザルトコード。キャラクタ型の配列 dFilename[90] には、起動したファイルの名前が返される。

$A357 TSEventAvail

引　数► word eventMask ; イベントマスク
long tsEventRecPtr ; タスクマンイベントレコードへのポインタ

返り値► D0.L リザルトコード

機　能► タスクマンのイベントキューから eventMask の各ビットで指定したイベントのうち、イベントキューのなかで最初に見つかったものを参照する。イベントの内容は、tsEventRecPtr で指定したタスクマンイベントレコードに返る。
イベントマンのイベントのほか、タスクマンのイベントもサポートされる。
このコールを呼ぶことでタスクの切り替えが発生する。
再配置が発生する。

Cの関数► int *TSEventAvail*(int eventMask, TsEvent *tsEventRecPtr);
返り値はリザルトコード。

$A358　TSGetEvent

引　数► word　eventMask　；イベントマスク
long　tsEventRecPtr　；タスクマンイベントレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► タスクマンのイベントキューから eventMask の各ビットで指定したイベントを取り出し、削除する。取り出したイベントの内容は、tsEventRecPtr で指定したタスクマンイベントレコードに返る。イベントマンのイベントのほか、タスクマンのイベントもサポートされる。
再配置が発生する。
Cの関数► int *TSGetEvent*(int eventMask, TsEvent *tsEventRecPtr);
返り値はリザルトコード。

$A35A　TSPostEventTsk

引　数► long　mes1　；メッセージ1
long　mes2　；メッセージ2
word　what2　；タスクマンイベントコード
byte　Hmode1　；ハンドルモード1
byte　Hmode2　；ハンドルモード2
返り値► D0.L　リザルトコード
機　能► mes1, mes2, what2 で指定したデータをそれぞれ whom, whom2, what2 に持つようなタスクマンイベントレコードを作成し、それをタスクマンのイベントキューに登録する。このイベントの種類は 12(E_SYSTEM1) に固定。
whom あるいは whom2 に格納されている引数がハンドルである場合、Hmode1 あるいは Hmode2 に 0 以外を指定すると、別のブロックを作成して whom, whom2 が指すブロックの内容をコピーする。タスクマンのイベントキューに登録されるタスクマンイベントレコードの whom, whom2 には新しく作成したブロックへのハンドルが格納される。この場合、タスクマンイベントレコードを廃棄する際、これらのブロックは同時に廃棄される。
再配置が発生する。
Cの関数► int *TSPostEventTsk*(long mes1, long mes2, int what2, BOOLEAN Hmode1, BOOLEAN Hmode2);
返り値はリザルトコード。

$A35B　TSGetTdb

引　数► long　buffPtr　；タスク管理レコードが返るバッファ(512 バイト)のアドレス
word　taskID　；タスク ID

返り値► D0.L　　リザルトコード

機　能► taskID で指定したタスクのタスク管理テーブルの内容を buffPtr で指定したバッファに返す。taskID として -1 を指定した場合、このコールを呼んだタスクのタスク管理レコードが返る。

Cの関数► int *TSGetTdb*(Task *buffPtr, int taskID);

返り値はリザルトコード。

$A35C　　TSSetTdb

引　数► long　　buffPtr　　; タスク管理レコードへのポインタ

word　　taskID　　; タスク ID

返り値► D0.L　　リザルトコード

機　能► taskID で指定したタスクのタスク管理テーブルに buffPtr で指定したタスク管理レコードの内容をコピーする。taskID として -1 を指定した場合、このコールを呼んだタスクのタスク管理テーブルにコピーする。

Cの関数► int *TSSetTdb*(Task *buffPtr, int taskID);

返り値はリザルトコード。

$A35E　　TSGetWindowPos

引　数► なし

返り値► D0.L　　ウィンドウをオープンするポイント（グローバル座標）

機　能► ウィンドウをオープンするポイントを得る。得られたポイントはウィンドウレクタングルの左上座標として指定する。

Cの関数► LPoint *TSGetWindowPos*(void);

返り値はポイント。

$A35F　　TSCommunicate

引　数► word　　listener　　; 宛先となるタスクの ID

long　　tsEventRecPtr　　; タスクマンイベントレコードへのポインタ

word　　mode　　; 返答モード

0	返事を受け付けない
1	返事を受け付ける

返り値► D0.L

0	正常終了
1	返事が届いた
-1	宛先のタスクが存在しない
-2	宛先のタスクの準備が整っていない
< 0	リザルトコード

機　能► listener で指定したタスクに tsEventRecPtr で指定したタスクマンイベ

ントレコードの内容のイベントを発生させる。すなわち、タスクマンイベントレコードの内容によって、タスク間通信を行う。これによって、宛先のタスクに発生するイベントは13(E_SYSTEM2)に固定。

タスクマンイベントレコードの what に入っているイベントコードは意味を持たない。what2 にはメッセージの意味を示すコード(128〜32767)を、tskid には自分のタスクIDを指定する。whom, whom2 には引数を、when には必要があればチックカウントを格納する。

メッセージを送られた側のタスクは、返事を返す場合、送られてきたタスクマンイベントレコードのなかに返事となるデータを格納する。

このコールは、他のタスクと通信中のタスクにも割り込んでメッセージを送ることができるため、混乱する場合があるので、通常は$A418 TSSendMes, $A417 TSAnswer を使用してタスク間通信を行う。

再配置が発生する。

Cの関数► int *TSCommunicate*(int listener, TsEvent *tsEventRecPtr, int mode);

返り値は結果を意味する数値。

$A360　　TSGetID

引　数► なし

返り値► D0.L　　タスクID

機　能► このコールを呼んだタスクのタスクIDを返す。

Cの関数► int *TSGetID*(void);

返り値はタスクID。

$A361　　TSMakeEvent

引　数►

long	mes1	;メッセージ1
long	mes2	;メッセージ2
word	what2	;タスクマンイベントコード
byte	Hmode1	;ハンドルモード1
byte	Hmode2	;ハンドルモード2
long	tsEventRecPtr	;タスクマンイベントレコードへのポインタ

返り値► D0.L　　リザルトコード

機　能► mes1, mes2, what2 で指定したデータをそれぞれ whom, whom2, what2 に持つようなタスクマンイベントレコードを、tsEventRecPtr で指定したレコードのなかに作成する。このイベントの種類は12(E_SYSTEM1)に固定。

whom あるいは whom2 に格納されている引数がハンドルである場合、Hmode1 あるいは Hmode2 に 0 以外を指定すると、別のブロックを作成して whom, whom2 が指すブロックの内容をコピーする。作成されるタスクマンイベントレ

コードの whom, whom2 には新しく作成したブロックへのハンドルが格納される。
再配置が発生する。

Cの関数► int *TSMakeEvent*(long mes1, long mes2, int what2, BOOLEAN Hmode1, BOOLEAN Hmode2, TsEvent *tsEventRecPtr);
返り値はリザルトコード。

$A364 TSGetStartMode

引　数► なし

返り値► D0.L　= 0　終了画面を保存しない
　　　　　　　≠ 0　終了画面を保存する

機　能► スタート画面の保存モードを返す。

Cの関数► BOOLEAN *TSGetStartMode*(void);
返り値はスタート画面の保存モード。

$A365 TSSetStartMode

引　数► word　mode　；スタート画面の保存モード
　　　　　　　　　　　= 0　終了画面を保存しない
　　　　　　　　　　　≠ 0　終了画面を保存する

返り値► なし

機　能► スタート画面の保存モードを mode に設定する。

Cの関数► void *TSSetStartMode*(int mode);
返り値はない。

$A367 TSOpen

引　数► long　namePtr　；ファイル名へのポインタ (ASCIIZ 型)
　　　　word　mode　；アクセスモード

返り値► D0.L　ファイルハンドル/リザルトコード

機　能► namePtr で指定した名前のファイルを、mode のアクセスモードでオープンする。DOS コールの$FF3D OPEN と同様の動作を行う。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
  ファイル名がクリーナに含まれるときはオープンできない。
- タスクマンのイベントが発生する

20(OPENFILE、“ファイルのオープン”)と16(CREATFILE、“ファイルの作成”)のどちらか、または両方が発生する。

Cの関数► int *TSOpen*(const char *namePtr, int mode);
返り値は、ファイルハンドルまたはリザルトコード。

$A368 TSClose

引　数► word fileID ;ファイルハンドル
返り値► D0.L リザルトコード
機　能► fileIDで指定されたファイルをクローズする。DOSコールの$FF3E CLOSEと同様の動作を行う。
再配置が発生する。
拡張されている点は次のとおり。
- タスクマンのイベントが発生する
  21(CLOSEFILE、“ファイルのクローズ”)が発生する。

Cの関数► int *TSClose*(int fileID);
返り値はリザルトコード。

$A369 TSRmDirH

引　数► long nameHdl ;ディレクトリ名へのハンドル(ASCIIZ型)
返り値► D0.L リザルトコード
機　能► nameHdlで指定したディレクトリを削除する。DOSコールの$FF3A RMDIRと同様の動作を行う。ディレクトリ名は必ずフルパスで指定する必要がある。
再配置が発生する。
拡張されている点は次のとおり。
- クリーナに対応している
- タスクマンのイベントが発生する
  10(RMDIR、“ディレクトリの削除”)が発生する。

Cの関数► int *TSRmDirH*(char **nameHdl);
返り値はリザルトコード。

$A36A TSCopyH

引　数► long sNameHdl ;コピー元のファイル名へのハンドル(ASCIIZ型)
long dNameHdl ;コピー先のファイル名へのハンドル(ASCIIZ型)
返り値► D0.L リザルトコード

機　能► sNameHdl で指定したファイルを、dNameHdl で指定したファイルにコピーする。ファイル名は必ずフルパスで指定する必要がある。
コピーが正常に終了した場合、タスクマンイベント 14(COPYFILE、“ファイルのコピー”) が発生する。
コピー先ファイルがクリーナに含まれる場合はクリーナがフラッシュされる。
再配置が発生する。

Cの関数► int *TSCopyH*(char **sNameHdl, char **dNameHdl);
返り値はリザルトコード。

$A36B　　TSMkDirH

引　数► long　　nameHdl　　;ディレクトリ名へのハンドル (ASCIIZ 型)

返り値► D0.L　　リザルトコード

機　能► nameHdl で指定したディレクトリを作成する。DOS コールの$FF39 MKDIR と同様の動作を行う。ディレクトリ名は必ずフルパスで指定する必要がある。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
  ディレクトリ名がクリーナに含まれる場合はクリーナがフラッシュされる。
- タスクマンのイベントが発生する
  15(MKDIR、“ディレクトリの作成”) が発生する。

Cの関数► int *TSMkDirH*(char **nameHdl);
返り値はリザルトコード。

$A36C　　TSMoveH

引　数► long　　sNameHdl　　;移動元のファイル名へのハンドル (ASCIIZ 型)
long　　dNameHdl　　;移動先のファイル名へのハンドル (ASCIIZ 型)

返り値► D0.L　　リザルトコード

機　能► sNameHdl で指定したファイルを、dNameHdl で指定したファイルに移動する。DOS コールの$FF56 RENAME と同様の動作を行う。同じパスにあるファイルの場合、リネームする。ファイル名は必ずフルパスで指定する必要がある。
正常に終了した場合、タスクマンイベント 13(MOVEFILE、“ファイルの移動”) が発生する。
移動先ファイルがクリーナに含まれる場合はクリーナがフラッシュされる。
再配置が発生する。

Cの関数► int *TSMoveH*(char **sNameHdl, char **dNameHdl);
返り値はリザルトコード。

$A36D　TSCreate

引　数► long　namePtr　; ファイル名へのポインタ（ASCIIZ 型）
　　　　word　atr　; アトリビュート

返り値► D0.L　ファイルハンドル/リザルトコード

機　能► namePtr で指定した名前のファイルを、atr のアトリビュートで新規作成する。DOS コールの$FF3C CREATE と同様の動作を行う。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
  ファイル名がクリーナに含まれるときはクリーナがフラッシュされる。
- タスクマンのイベントが発生する
  16(CREATFILE、“ファイルの作成”) が発生する。

Cの関数► int *TSCreate*(const char *namePtr, int atr);
返り値はファイルハンドルまたはリザルトコード。

$A36E　TSDeleteH

引　数► long　nameHdl　; ファイル名へのハンドル（ASCIIZ 型）

返り値► D0.L　リザルトコード

機　能► nameHdl で指定したファイルを実際に削除する。DOS コールの$FF41 DELETE と同様の動作を行う。ファイル名は必ずフルパスで指定する必要がある。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
- タスクマンのイベントが発生する
  11(DELETEFILE、“ファイルの削除”) が発生する。

Cの関数► int *TSDeleteH*(char **nameHdl);
返り値はリザルトコード。

$A36F　TSTrash

引　数► long　sNameHdl　; 移動元のファイル名へのハンドル（ASCIIZ 型）
　　　　long　dNameHdl　; 移動先のファイル名へのハンドル（ASCIIZ 型）

返り値► D0.L　リザルトコード

機　能► sNameHdl で指定したファイルを、dNameHdl で指定したファイルに移動する。クリーナから、またはクリーナへの移動を行い、実際にファイルが移動されるわけではない。ファイル名は必ずフルパスで指定する必要がある。

正常に終了した場合、タスクマンイベント13(MOVEFILE、“ファイルの移動”)が発生する。
再配置が発生する。

Cの関数► int *TSTrash*(char **sNameHdl, char **dNameHdl);
返り値はリザルトコード。

$A370 TSFiles

引　数► long fileBuff ;結果が返るバッファのアドレス
long namePtr ;ファイル名へのポインタ (ASCIIZ 型)
word atr ;アトリビュート

返り値► D0.L リザルトコード

機　能► namePtr で指定したファイル名と atr で指定したアトリビュートに合致するような最初のファイルを探し、そのファイルについての情報を fileBuff に返す。DOS コールの$FF4E FILES と同様の動作を行う。
クリーナに入っていることになっているファイルは検索の対象とはならない。
再配置が発生する。

Cの関数► int *TSFiles*(void *fileBuff,const char *namePtr,int atr);
返り値はリザルトコード。

$A371 TSNFiles

引　数► long fileBuff ;結果が返るバッファのアドレス
long namePtr ;ファイル名へのポインタ (ASCIIZ 型)
word atr ;アトリビュート

返り値► D0.L リザルトコード

機　能► 前回の TSFiles/TSNFiles に続くようなファイルを探し、そのファイルについての情報を fileBuff に返す。DOS コールの$FF4F NFILES と同様の動作を行う。
クリーナに入っていることになっているファイルは検索の対象とはならない。
TSNFiles を呼んだあとにコールし、namePtr と atr は TSNFiles と同じものを指定する。
再配置が発生する。

Cの関数► int *TSNFiles*(void *fileBuff,const char *namePtr,int atr);
返り値はリザルトコード。

$A372 TSCopyP

引　数► long sNamePtr ;コピー元のファイル名へのポインタ (ASCIIZ 型)

long　dNamePtr　；コピー先のファイル名へのポインタ（ASCIIZ型）

返り値► D0.L　リザルトコード

機　能► sNamePtrで指定したファイルをdNamePtrで指定したファイルにコピーする。ファイル名は必ずフルパスで指定する必要がある。
コピーが正常に終了した場合、タスクマンイベント14(COPYFILE、“ファイルのコピー”)が発生する。
コピー先ファイルがクリーナに含まれる場合はクリーナがフラッシュされる。
再配置が発生する。

Cの関数► int *TSCopyP*(const char *sNamePtr,const char *dNamePtr);
返り値はリザルトコード。

$A373　TSDeleteP

引　数► long　namePtr　；ファイル名へのポインタ（ASCIIZ型）

返り値► D0.L　リザルトコード

機　能► namePtrで指定したファイルを実際に削除する。DOSコールの$FF41 DELETEと同様の動作を行う。ファイル名は必ずフルパスで指定する必要がある。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
- タスクマンのイベントが発生する
11(DELETEFILE、“ファイルの削除”)が発生する。

Cの関数► int *TSDeleteP*(const char *namePtr);
返り値はリザルトコード。

$A374　TSRmDirP

引　数► long　namePtr　；ディレクトリ名へのポインタ（ASCIIZ型）

返り値► D0.L　リザルトコード

機　能► namePtrで指定したディレクトリを削除する。DOSコールの$FF3A RMDIRと同様の動作を行う。ディレクトリ名は必ずフルパスで指定する必要がある。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
- タスクマンのイベントが発生する
10(RMDIR、“ディレクトリの削除”)が発生する。

Cの関数► int *TSRmDirP*(const char *namePtr);
返り値はリザルトコード。

$A375　TSMkDirP

引　数► long　namePtr　；ディレクトリ名へのポインタ（ASCIIZ型）

返り値► D0.L　リザルトコード

機　能► namePtr で指定したディレクトリを作成する。DOS コールの$FF39 MKDIR と同様の動作を行う。ディレクトリ名は必ずフルパスで指定する必要がある。
再配置が発生する。
拡張されている点は次のとおり。

- クリーナに対応している
  ディレクトリ名がクリーナに含まれる場合は、クリーナがフラッシュされる。
- タスクマンのイベントが発生する
  15(MKDIR、"ディレクトリの作成") が発生する。

Cの関数► int *TSMkDirP*(const char *namePtr);
返り値はリザルトコード。

$A376　TSMoveP

引　数► long　sNamePtr　；移動元のファイル名へのポインタ（ASCIIZ型）
long　dNamePtr　；移動先のファイル名へのポインタ（ASCIIZ型）

返り値► D0.L　リザルトコード

機　能► sNamePtr で指定したファイルを dNamePtr で指定したファイルに移動する。DOS コールの$FF56 RENAME と同様の動作を行う。同じパスにあるファイルの場合、リネームする。ファイル名は必ずフルパスで指定する必要がある。
正常に終了した場合、タスクマンイベント 13(MOVEFILE、"ファイルの移動") が発生する。
移動先ファイルがクリーナに含まれる場合は、クリーナがフラッシュされる。
再配置が発生する。

Cの関数► int *TSMoveP*(const char *sNamePtr,const char *dNamePtr);
返り値はリザルトコード。

$A378　TSChMod

引　数► long　namePtr　；ファイル名へのポインタ（ASCIIZ 型）
word　atr　；アトリビュート

返り値► D0.L　リザルトコード

機　能► namePtr で指定したファイルのアトリビュートを atr に変更する。ファイル名は必ずフルパスで指定する必要がある。
正常に終了した場合、タスクマンイベント 22(CHMODFILE、"ファイルの移動")

が発生する。
再配置が発生する。

Cの関数► int *TSChMod*(const char *namePtr, int atr);
返り値はリザルトコード。

$A379　TSWhatFile

引　数► long　tsEventRecPtr　;タスクマンイベントレコードへのポインタ
long　namePtr　;ファイル名へのポインタ（ASCIIZ 型）

返り値► D0.L　上位ワード

上位バイト	
0	ソース側
1	ディスティネーション側
下位バイト	
0	namePtr に対して
1	namePtr を含むパスに対して
2	namePtr に含まれるファイルに対して
4	namePtr に含まれるサブディレクトリ内のファイルに対して

下位ワード

0	無関係
1	削除される
2	削除された
3	移動した
4	イジェクトされる
5	イジェクトされた
6	変更があった（Create も含む）
7	オープンされた
8	クローズされた
9	指定したパスのクローズ
10	ボリューム名の削除
11	ボリューム名の作成

機　能► tsEventRecPtr で指定したタスクマンイベントの内容が、namePtr で指定したファイルに関係があるかどうかを調べる。ファイル名は必ずフルパスで指定する必要がある。
対応するタスクマンイベントは 3(NOTICEEJECT、"ディスクイジェクトの予告") ～ 30(CLOSE、"指定したパスのクローズ")、51(EMPTYTRASH、"クリーナのフラッシュ")。

Cの関数► long *TSWhatFile*(TsEvent *tsEventRecPtr,
const char *namePtr);
返り値は結果を意味する数値。

$A37B　TSDeleteVoname

引　数► long　namePtr　;ボリューム名へのポインタ (ASCIIZ 型)
返り値► D0.L　リザルトコード
機　能► namePtr で指定したボリューム名を削除する。必ずフルパスで指定する必要がある。
正常に終了した場合、タスクマンイベント 12(DELETEVONAME、“ボリューム名の削除”) が発生する。
再配置が発生する。
Cの関数► int *TSDeleteVoname*(const char *namePtr);
返り値はリザルトコード。

$A37C　TSCreateVoname

引　数► long　namePtr　;ボリューム名へのポインタ (ASCIIZ 型)
返り値► D0.L　リザルトコード
機　能► namePtr で指定したボリューム名を作成する。必ずフルパスで指定する必要がある。
正常に終了した場合、タスクマンイベント 17(CREATEVONAME、“ボリューム名の作成”) が発生する。
再配置が発生する。
Cの関数► int *TSCreateVoname*(const char *namePtr);

$A381　TSSearchFileND

引　数► long　sNamePtr　;ファイル名へのポインタ (ASCIIZ 型)
long　dNamePtr　;発見したファイルの名前が返るバッファ(90 バイト) のアドレス
long　path　;パス名へのポインタ (ASCIIZ 型)
返り値► D0.L　ファイルサイズ/リザルトコード
機　能► sNamePtr で指定したファイルを検索し、発見できたファイルの名前を dNamePtr に返す。検索は、sNamePtr のなかにパスが含まれる場合はそのパスから、そうでない場合は path で指定したパスから開始する。そこで見つからなかった場合は、ドライブ A のルートディレクトリから順に検索していく。
検索中のダイアログは表示しない。
dNamePtr は sNamePtr と同じものを指定してもよい。
再配置が発生する。
Cの関数► int *TSSearchFileND*(const char *sNamePtr, char *dNamePtr, const char *path);
返り値はファイルサイズまたはリザルトコード。

$A386　TSGetOpen

引　数► なし
返り値► D0.L　オープン中のファイル数/リザルトコード
　　　　A0.L　オープンファイル名管理レコードの配列へのハンドル
機　能► 現在オープン中のファイル数と、オープンファイル名管理レコードの配列へのハンドルを返す。
Cの関数► int *TSGetOpen*(OpenFile ***openFileList);
返り値は、オープン中のファイル数またはリザルトコード。
OpenFile 型のハンドル openFileList には、オープンファイル名管理レコードの配列へのハンドルが格納される。

$A387　TSZeroDrag

引　数► なし
返り値► D0.L　リザルトコード
機　能► ドラッグバッファの内容をクリアする。
再配置が発生する。
Cの関数► int *TSZeroDrag*(void);
返り値はリザルトコード。

$A388　TSPutDrag

引　数► long　celLength　；セルリストのサイズ
　　　　long　celHdl　；セルリストへのハンドル
返り値► D0.L　リザルトコード
機　能► ドラッグバッファに celHdl で指定した celLength のサイズのセルリストを追加する。
疑似ハンドルも可。
再配置が発生する。
Cの関数► int *TSPutDrag*(long celLength, Handle celHdl);
返り値はリザルトコード。

$A389　TSGetDrag

引　数► なし
返り値► D0.L　= 0　正常終了
　　　　　　　≠ 0　ドラッグデータがない
　　　　A0.L　ドラッグレコードへのポインタ

機　能► システム内のドラッグレコードへのポインタを返す。
Cの関数► int *TSGetDrag*(Drag **drag);
返り値は結果を意味する数値。
Drag 型のポインタ drag には、ドラッグレコードへのポインタが格納される。

$A38A　TSBeginDrag

引　数► long　pt　; ドラッグを開始するポイント（グローバル座標）
返り値► D0.L　リザルトコード
機　能► pt で指定した位置からドラッグを開始する。ビットイメージのデータが作成される。
あらかじめ、グラフポートをセットしておく必要がある。
再配置が発生する。
Cの関数► int *TSBeginDrag*(LPoint pt);
返り値はリザルトコード。

$A38C　TSEndDrag

引　数► word　mode　= 0　普通に終了する
≠ 0　ラバーバンドを元の位置までアニメーションさせて終了する
返り値► D0.L　リザルトコード
機　能► ドラッグを終了する。ビットイメージのデータは開放される。
再配置が発生する。
Cの関数► int *TSEndDrag*(int mode);
返り値はリザルトコード。

$A38D　TSHideDrag

引　数► なし
返り値► D0.L　リザルトコード
機　能► 一時的にラバーバンドを消す。すでに消えている場合は何もしない。
再配置が発生する。
Cの関数► int *TSHideDrag*(void);
返り値はリザルトコード。

$A38E　　TSShowDrag

引　数► なし

返り値► D0.L　　リザルトコード

機　能► ラバーバンドを描画する。すでに描画されている場合は何もしない。
再配置が発生する。

Cの関数► int *TSShowDrag*(void);
返り値はリザルトコード。

$A38F　　TSZeroScrap

引　数► なし

返り値► D0.L　　リザルトコード

機　能► デスクトップスクラップバッファの内容をクリアする。
正常に終了した場合、タスクマンイベント50(TOSCRAP、"スクラップバッファへのデータ転送")が発生する。
再配置が発生する。

Cの関数► int *TSZeroScrap*(void);
返り値はリザルトコード。

$A390　　TSPutScrap

引　数► long　　celLength　　;セルリストのサイズ
long　　celHdl　　;セルリストへのハンドル

返り値► D0.L　　リザルトコード

機　能► デスクトップスクラップバッファに celHdl で指定した celLength のサイズのセルリストをセットする(追加ではなく置換)。正常に終了した場合、タスクマンイベント50(TOSCRAP、"スクラップバッファへのデータ転送")が発生する。
疑似ハンドルも可。
再配置が発生する。

Cの関数► int *TSPutScrap*(long celLength, Handle celHdl);
返り値はリザルトコード。

$A391　　TSGetScrap

引　数► なし

返り値► D0.L　　デスクトップスクラップバッファのサイズ
/リザルトコード

A0.L　デスクトップスクラップバッファへのポインタ

機　能► システム内のデスクトップスクラップバッファへのポインタを返す。

Cの関数► long *TSGetScrap*(Scrap **scrap);

返り値は、デスクトップスクラップバッファのサイズまたはリザルトコード。Scrap 型のポインタ scrap には、デスクトップスクラップバッファへのポインタが格納される。

$A397　TSSearchTrashpath

引　数► long　nameHdl　；ファイル名へのハンドル（ASCIIZ 型）

返り値► D0.L　= 0　発見できた

≠ 0　発見できなかった

機　能► nameHdl で指定したファイルがクリーナのなかにあるかどうかを調べる。ファイル名はフルパスで指定する必要がある。

Cの関数► int *TSSearchTrashpath*(char **nameHdl);

返り値は結果を意味する数値。

$A398　TSSearchTrashfile

引　数► long　nameHdl　；ファイル名へのハンドル（ASCIIZ 型）

返り値► D0.L　= 0　発見できた

≠ 0　発見できなかった

機　能► nameHdl で指定したファイルがクリーナのなかにあるかどうかを調べる。ファイルが含まれるディレクトリがクリーナのなかにあった場合も、クリーナに含まれることになる。ファイル名はフルパスで指定する必要がある。

Cの関数► int *TSSearchTrashfile*(char **nameHdl);

返り値は結果を意味する数値。

$A399　TSEmptyTrash

引　数► なし

返り値► D0.L　リザルトコード

機　能► クリーナに含まれるすべてのファイルを削除し、クリーナバッファをフラッシュする。正常に終了した場合、タスクマンイベント 51(EMPTYTRASH、“クリーナのフラッシュ”) が発生する。

再配置が発生する。

Cの関数► int *TSEmptyTrash*(void);

返り値はリザルトコード。

タスクマン

$A39B　TSSearchdpb

引　数►	word	mediabyte	;メディアバイト
	word	unitNo	;ユニット番号
返り値►	D0.L	ドライブ番号/リザルトコード	
	A0.L	該当するドライブの DPB のアドレス	

機　能► mediabyte で指定したメディアバイトと unitNo で指定したユニット番号を持つドライブを探し、発見できた場合はそのドライブ番号と DPB のアドレスを返す。

Cの関数► int *TSSearchdpb*(int mediabyte, int unitNo);
返り値は、ドライブ番号またはリザルトコード。
DPB へのポインタを得ることはできない。

$A39D　TSDrvctrl

引　数► word　mode　;動作モード

0	状態を返すだけ
1	イジェクトする
2	イジェクトを禁止する
3	イジェクトを許可する
4	LED 点滅モードに設定
5	LED 点滅モードを解除

word　drvNo　;ドライブ番号

0	クリーナ
1	ドライブ A
	⋮

返り値► D0.L　ドライブの状態/リザルトコード

機　能► drvNo で指定したドライブに対して、mode で指定したコントロールを行う。mode = 0 の場合、ドライブの状態が返される。DOS コールの$FF0F DRVCTRL と同様の動作を行う。
実際にドライブへのアクセスは行わず、システム内部に記憶されている状態を返す。内部に記憶されている状態の更新は、ドライブの挿入/イジェクト時に行われる。

Cの関数► int *TSDrvctrl*(int mode, int drvNo);
返り値は、ドライブの状態またはリザルトコード。

$A39E　TSDrvctrl2

引　数► word　mode　;動作モード

0	状態を返すだけ
1	イジェクトする
2	イジェクトを禁止する
3	イジェクトを許可する
4	LED 点滅モードに設定
5	LED 点滅モードを解除

word　drvNo　; ドライブ番号

0	クリーナ
1	ドライブ A
	⋮

返り値► D0.L　ドライブの状態/リザルトコード

機　能► drvNo で指定したドライブに対して、mode で指定したコントロールを行う。ドライブの状態は mode = 0 の場合に返される。DOS コールの$FF0F DRVCTRL と同様の動作を行う。
実際にドライブへのアクセスを行い、さらにフォーマットのチェックも行う。フォーマットが Human68k のフォーマットと異なる場合、ノットレディのビットを立てる。

Cの関数► int *TSDrvctrl2*(int mode, int drvNo);
返り値は、ドライブの状態またはリザルトコード。

$A3A2　SXCallWindM

引　数► long　winPtr　; ウィンドウレコードへのポインタ
long　tsEventRecPtr　; タスクマンイベントレコードへのポインタ

返り値► D0.L　ウィンドウのパートコード/リザルトコード

機　能► tsEventRecPtr で指定したイベントに対応した処理を winPtr で指定したウィンドウに対して行う。
再配置が発生する。
サポートされる処理は以下のとおり。

- ウィンドウの移動
- ウィンドウサイズの変更
- ウィンドウのズーム
- クリップの ON/OFF
- ウィンドウの付属品（クローズボックス、矢印）などのコントロール

Cの関数► int *SXCallWindM*(Window *winPtr, TsEvent *tsEventRecPtr);
返り値は、ウィンドウのパートコードまたはリザルトコード。

$A3A3　SXCallCtrlM

引　数► long　winPtr　; ウィンドウレコードへのポインタ

long　tsEventRecPtr　; タスクマンイベントレコードへのポインタ
long　ctrlHdlV　; 垂直スクロールバーへのハンドル（省略の場合 0）
long　ctrlHdlH　; 水平スクロールバーへのハンドル（省略の場合 0）
long　dRectPtr　; スクロールを行う領域を示すレクタングルへのポインタ

返り値► D0.L　パートコード/リザルトコード
A0.L　コントロールへのハンドル

機　能► tsEventRecPtr で指定したイベントに対応した処理を winPtr で指定したウィンドウに属するコントロールに対して行う。
ctrlHdlV, ctrlHdlH には、ウィンドウ内部をスクロールさせるために置いた垂直、水平各スクロールバーへのハンドルを指定する。dRectPtr には、スクロールを行う領域を示すレクタングルへの（疑似）ポインタを指定する。これによって、指定した範囲を超えるようなスクロールは行わないようになる。
ctrlHdlV, ctrlHdlH, dRectPtr は、スクロールバーの処理が不要ならば省略可。
再配置が発生する。
サポートされる処理は以下のとおり。

- セレクトボタン/オルタネートボタンの ON/OFF
- スクロールバーによるスクロール

Cの関数► int *SXCallCtrlM*(Window *winPtr, TsEvent *tsEventRecPtr, Control **ctrlHdlV, Control **ctrlHdlH, Rect *dRectPtr, Control ***ctrl);
返り値は、コントロールのパートコードまたはリザルトコード。
Control 型のハンドル ctrl には、操作されたコントロールのハンドルが格納される。

$A3AA　SXInvalScBar

引　数► long　winPtr　; ウィンドウレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► winPtr で指定したウィンドウレコードの垂直・水平スクロールバーの描画される部分をアップデートリージョンに加える。
再配置が発生する。

Cの関数► int *SXInvalScBar*(Window *winPtr);
返り値はリザルトコード。

$A3AB　SXValidScBar

引　数► long　winPtr　；ウィンドウレコードへのポインタ
返り値► D0.L　リザルトコード
機　能► winPtr で指定したウィンドウレコードの垂直・水平スクロールバーの描画される部分をアップデートリージョンから除く。
再配置が発生する。
Cの関数► int *SXValidScBar*(Window *winPtr);
返り値はリザルトコード。

$A3BB　TSISRecToStr

引　数► long　ISRecPtr　；アイコン管理レコードへのポインタ
long　dstr　；フルパス名が返るバッファ(90 バイト）のアドレス
返り値► D0.L　パス名の長さ/リザルトコード
A0.L　パス名の終端 ＋1
機　能► ISRecPtr で指定したアイコン管理レコードからフルパス名を作成し、dstr に返す。
Cの関数► int *TSISRecToStr*(IcState *ISRecPtr, char *dstr);
返り値は、パス名の長さまたはリザルトコード。

$A3BF　TSCreateISFile

引　数► long　nameHdl　；ファイル名へのハンドル（ASCIIZ 型）
long　ISRecPtr　；アイコン管理レコードへのポインタ
返り値► D0.L　ファイルの種類

0	ファイルまたはサブディレクトリ
1	クリーナのルートディレクトリ
2	クリーナのファイルまたはディレクトリ
3	ルートディレクトリ

A0.L　アイコン管理レコードへのポインタ
機　能► nameHdl で指定したファイル名をもとに ISRecPtr で指定したアドレスからアイコン管理レコードを作成する。ファイル名はフルパスで指定する必要がある。
再配置が発生する。
Cの関数► int *TSCreateISFile*(char **nameHdl, IcState *ISRecPtr);
返り値は結果を意味する数値。

$A3CC　SXFileConnPath

引　数► long　namePtr　；ファイル名へのポインタ（ASCIIZ 型）
long　pathPtr　；パス名へのポインタ（ASCIIZ 型）

返り値► D0.L　= 0　ファイルのパスは pathPtr と等しくない
≠ 0　ファイルのパスは pathPtr と等しい

機　能► namePtr で指定したファイルのパスが、pathPtr で指定したパスと等しいかどうかを調べる。ファイル名、パス名はフルパスで指定する必要がある。

Cの関数► BOOLEAN *SXFileConnPath*(const char *namePtr, const char *pathPtr);
返り値は結果を意味する数値。

$A3CD　SXFileInPath

引　数► long　namePtr　；ファイル名へのポインタ（ASCIIZ 型）
long　pathPtr　；パス名へのポインタ（ASCIIZ 型）

返り値► D0.L

0	ファイルのパスと pathPtr は等しい
1	ファイルは pathPtr 上にある
-1	ファイルは pathPtr 上にない

機　能► namePtr で指定したファイルが pathPtr で指定したパスに含まれるかどうかを調べる。ファイル名、パス名はフルパスで指定する必要がある。

Cの関数► int *SXFileInPath*(const char *namePtr,const char *pathPtr);
返り値は結果を意味する数値。

$A3D0　SXFnamecmp

引　数► long　namePtr　；ファイル名へのポインタ（ASCIIZ 型）
word　nameLength　；名前部分の長さ（通常は 0 を指定）
long　maskPtr　；マスク文字列へのポインタ（ASCIIZ 型）
word　maskLength　；マスク文字列の名前部分の長さ（通常は 0 を指定）

返り値► D0.L

0	完全にマッチした
1	名前と拡張子のどちらかで“*”を使用してマッチ
2	名前と拡張子のどちらか一方が“*”のみでマッチ、または、名前拡張子両方で“*”を使用してマッチ
3	名前と拡張子の両方で“*”を使用し、かつ、どちらか一方が“*”のみでマッチ
4	“*.*”でマッチ
-1	マッチしない

機　能► namePtr で指定したファイル名と maskPtr で指定したマスク文字列がマッチするかどうかを調べる。大文字と小文字は区別される。

Cの関数► int *SXFnamecmp*(const char *namePtr, int nameLength,
const char *maskPtr, int maskLength);
返り値は結果を意味する数値。

$A3D4 SXSearchFname

引　数► long namePtr ；ファイル名へのポインタ（ASCIIZ 型）
返り値► D0.L 拡張子
A0.L ファイル名の先頭
機　能► namePtr で指定したフルパスのファイル名のなかの、ファイル名が始まる位置を返す。
Cの関数► long *SXSearchFname*(const char *namePtr, char **top);
返り値は拡張子。
char 型のポインタ top には、ファイル名の先頭位置が格納される。

$A3D8 SXStoLower

引　数► long strPtr ；文字列へのポインタ
word length ；変換するバイト数
返り値► なし
機　能► strPtr で指定した文字列を length バイトだけ小文字に変換する。

$A3D9 SXStoUpper

引　数► long strPtr ；文字列へのポインタ
word length ；変換するバイト数
返り値► なし
機　能► strPtr で指定した文字列を length バイトだけ大文字に変換する。

$A3DA SXStoUpper2

引　数► long bufPtr ；結果が返るバッファのアドレス
long strPtr ；文字列へのポインタ
word length ；変換するバイト数
返り値► なし
機　能► strPtr で指定した文字列を length バイトだけ大文字に変換し、bufPtr で指定されたバッファに返す。

$A3E9　　SXVer

引　数► なし

返り値► D0.L　　SX システムのバージョン番号

機　能► SX-WINDOW のバージョン番号を返す。ver.3.10 ならば、返り値は 0x0310 となる。

Cの関数► int *SXVer*(void);
返り値はバージョン番号。

$A3EA　　TSTakeParam

引　数►

long	commandPtr	;コマンドライン文字列（LASCII 型）へのポインタ
long	wRectPtr	;レクタングルレコードが返るバッファのアドレス
long	namePtr	;文字列が返るバッファ（90 バイト）のアドレス
word	mode	;argc, argv 保存モード

0	argc, argv を保存しない
1	argc, argv のブロックを作成して保存
2	argc, argv を destBuff に保存

long	destBuff	;argc, argv を保存するバッファ

返り値►

D0.L	bit0	ウィンドウ位置指定があった
	bit1	文字列があった
		/リザルトコード
A0.L	argc, argv を保存したバッファへのポインタ	

機　能► commandPtr で指定されたコマンドライン文字列を解析する。
ウィンドウ位置の指定オプション（“-Wx0,y0,x1,y1”）があった場合は、wRectPtr で指定されたバッファにレクタングルレコードを返す。
ウィンドウ位置の指定オプション以外の文字列は要素に分けられ、argc(要素数)、argv(要素の文字列) のテーブルが作成される。
再配置が発生する。

- mode = 0 の場合 :保存しない
  作成したテーブルは処理の終了時に廃棄され、アプリケーションには返されない。
  destBuff は意味を持たない。
- mode = 1 の場合 :ブロックを作成して保存
  再配置不能ブロックが作成され、そのなかにテーブルが作成される。アプリケーションには、そのブロックへのポインタが返される。
  destBuff は意味を持たない。

- mode = 2 の場合 :destBuff に保存
  destBuff で指定したバッファにテーブルが作成される。

いずれの場合も、最後の要素の文字列が namePtr で指定されたバッファに返る。

テーブルの内容は次のとおり。

+$00.L　要素数 (argc)
+$04.L　最初の要素へのポインタ (argv[1])
+$08.L　2 番目の要素へのポインタ (argv[2])
⋮

Cの関数► int *TSTakeParam*(const _LASCII commandPtr,Rect *wRectPtr, char *namePtr, int mode, char **destBuff, char **argvc);
返り値は、結果を意味するフラグまたはリザルトコード。
char 型のポインタ argvc には argv, argc を格納するために確保したブロックへのポインタが格納される。

$A3F4　TSFindTskn

引　数► long　namePtr　; タスク名を示す文字列 (ASCIIZ 型) へのポインタ
word　taskID　; タスク ID

返り値► D0.L　タスク ID(下位ワード)/リザルトコード

機　能► 現在登録されているタスクのうち、taskID より大きな ID を持ち、namePtr で指定したタスク名と一致するものを検索し、最初にみつかったもののタスク ID を返す。
タスク名にはワイルドカードを使用することが可能。
taskID として -1 を指定した場合、タスク ID に無関係に検索する。

Cの関数► long *TSFindTskn*(const char *namePtr, int taskID);
返り値は、タスク ID またはリザルトコード。

$A3F7　TSDriveCheckAll

引　数► word　mode　; ドライブをチェックするモード

0	挿入されていないドライブだけ調べる
≠ 0	すべてのドライブを調べる

返り値► D0.L　リザルトコード

機　能► ドライブを調べ、それまでに挿入/イジェクトが発生していたら、イベントを発生させる。
再配置が発生する。

Cの関数► int *TSDriveCheckAll*(int mode);

返り値はリザルトコード。

$A3F8　TSDriveCheck

引　数► word　drvNo　；ドライブ番号
返り値► D0.L　ドライブの状態/リザルトコード
機　能► drvNo で指定したドライブを調べ、それまでに挿入/イジェクトが発生していたら、イベントを発生させる。返り値のドライブの状態は、$A39D TSDrvctrl などと同様、DOS コールの$FF0F DRVCTRL の返り値と同様の意味を持つ。
再配置が発生する。
Cの関数► int *TSDriveCheck*(int drvNo);
ドライブの状態を意味する数値、またはリザルトコード。

$A3F9　TSISRecToExec

引　数► long　ISRecPtr　；アイコン管理レコードへのポインタ
long　namePtr　；実行ファイル名が返るバッファ(90 バイト)のアドレス
long　commandPtr　；コマンドライン文字列（LASCII 型）が返るバッファ(256 バイト)のアドレス
返り値► D0.L　リザルトコード
機　能► ISRecPtr で指定したアイコン管理レコードから起動するファイルの名前とコマンドライン文字列を作成し、それぞれ namePtr, commandPtr で指定されたバッファに返す。
再配置が発生する。
Cの関数► int *TSISRecToExec*(IcState *ISRecPtr, char *namePtr, _LASCII commandPtr);
返り値はリザルトコード。

$A3FA　TSGetDtopMode

引　数► なし
返り値► D0.L　= 0　デスクトップを保存しない
≠ 0　デスクトップをつねに保存する
機　能► 画面状態保存モードを返す。
Cの関数► BOOLEAN *TSGetDtopMode*(void);
返り値は結果を意味する数値。

$A3FB　　TSSetDtopMode

引　数► word　mode　; 画面状態保存モード

0	デスクトップを保存しない
≠ 0	デスクトップをつねに保存する

返り値► なし

機　能► 画面状態保存モードを mode に設定する。

Cの関数► void *TSSetDtopMode*(int mode);
返り値はない。

$A3FC　　TSSearchOpen

引　数► long　nameHdl　; ファイル名へのハンドル (ASCIIZ 型)
word　mode　; アクセスモード
word　fileID　; ファイルハンドル

返り値► D0.L

0	発見できた
-1	発見できなかった

A0.L　該当するオープンファイル名管理レコードへのポインタ

機　能► nameHdl の名前で、mode のアクセスモードでオープンされていて fileID のファイルハンドルを持つファイルが存在するかどうかを調べる。発見できた場合は、該当するオープンファイル名管理レコードのポインタが返る。このポインタは再配置可能ブロック内を指しているので、再配置が発生した場合は無効になることに注意。

Cの関数► int *TSSearchOpen*(char **nameHdl, int mode, int fileID, OpenFile **open);
返り値は、結果を意味する数値。
OpenFile 型のポインタ open には、オープンファイル名管理レコードへのポインタが返る。

$A3FE　　TSFindOwn

引　数► なし

返り値► D0.L　タスク ID (下位ワード)/リザルトコード

機　能► 現在登録されているタスクのうち、このコールを呼んだタスクと同じ名前のタスクを検索し、最初にみつかったもののタスク ID を返す。

Cの関数► long *TSFindOwn*(void);
返り値は、タスク ID またはリザルトコード。

$A3FF　TSCommunicateS

引　数► word　listener　; 宛先となるタスクの ID
long　tsEventRecPtr　; タスクマンイベントレコードへのポインタ
word　mode　; 返答モード

0	返事を受け付けない
1	返事を受け付ける

返り値► D0.L

0	正常終了
1	返事が届いた
-1	宛先のタスクが存在しない
-2	宛先のタスクの準備が整っていない
< 0	リザルトコード

機　能► listener で指定したタスクに tsEventRecPtr で指定したタスクマンイベントレコードの内容のイベントを発生させる。すなわち、タスクマンイベントレコードの内容によってタスク間通信を行う。宛先のタスクに発生させるイベントは、イベントレコード中に自由に設定できる。
スーパユーザ専用であり、タスク ID が 0（つまり、シェル）のタスク以外は使用してはならない。
再配置が発生する。

Cの関数► int *TSCommunicateS*(int listener, TsEvent *tsEventRecPtr, int mode);
返り値は、結果を意味する数値またはリザルトコード。

$A402　TSSearchFile2

引　数► byte　mode1　; 検索モード 1

0	ダイアログを表示しない
≠ 0	mode1 回以上探したら、ダイアログを表示する

byte　mode2　; 検索モード 2

0	1 個ファイルをみつけて終了
1	複数検索 ： 最初の検索
2	複数検索 ： 2 回目以降の検索
3	複数検索 ： 終了処理
4	複数検索 ： ダイアログのアップデート

long　SFRecPtr　; ファイル検索レコード（268 バイト）へのポインタ
long　sNamePtr　; ファイル名（ASCIIZ 型）へのポインタ
long　dNamePtr　; 該当ファイル名が返るバッファ（90 バイト）のアドレス
long　pathPtr　; カレントパス名（ASCIIZ 型）へのポインタ

返り値► D0.L　　ファイルサイズ/リザルトコード

機　能► sNamePtr で指定したファイルを検索し、発見できたファイルの名前を dNamePtr に返す。検索は、sNamePtr のなかにパスが含まれる場合はそのパスから、そうでない場合は pathPtr で指定したパスから開始する。そこで見つからなかった場合は、ドライブ A のルートディレクトリから順に検索していく。mode1 が 0 以外の場合、mode1 で示される回数だけ探したところでダイアログを表示する。

sNamePtr は dNamePtr と同じものを指定してもよい。

複数個のファイルを検索する機能が用意されている。

再配置が発生する。

- mode2 = 0 の場合 ： 1 個ファイルをみつけて終了
  sNamePtr での指定にマッチするようなファイルの最初の 1 つを発見したら終了する。つまり、$A403 TSSearchFile と同様。SFRecPtr は意味を持たない。
- mode2 = 1 の場合 ： 複数検索 ： 最初の検索
  複数個のファイルを検索するための準備（ファイル検索レコードの初期化）を行い、sNamePtr での指定にマッチするようなファイルのうち最初に見つかったものを返す。1 つファイルを発見するか、1 ドライブ探したところで帰ってくる。
- mode2 = 2 の場合 ： 複数検索 ： 2 回目以降の検索
  mode2 = 1 で行った処理の続きを行う。
- mode2 = 3 の場合 ： 複数検索 ： 終了処理
  複数個のファイルの検索の終了処理を行う。
- mode2 = 4 の場合 ： 複数検索 ： ダイアログのアップデート
  ダイアログを再描画する。複数検索を行っている途中でアップデートイベントが発生した場合のため。

Cの関数► int *TSSearchFile2*(BOOLEAN mode1,int mode2,void *SFRecPtr, const char *sNamePtr,char *dNamePtr,const char *pathPtr);
返り値は、ファイルサイズまたはリザルトコード。

$A403　TSSearchFile

引　数► long　sNamePtr　　; ファイル名（ASCIIZ 型）へのポインタ
long　dNamePtr　　; 発見したファイルの名前が返るバッファ（90 バイト）のアドレス
long　pathPtr　　; パス名へのポインタ（ASCIIZ 型）

返り値► D0.L　　ファイルサイズ/リザルトコード

機　能► sNamePtr で指定したファイルを検索し、発見できたファイルの名前を dNamePtr に返す。検索は、sNamePtr のなかにパスが含まれる場合はそのパスから、そうでない場合は pathPtr で指定したパスから開始する。そこで見つからなかった場合は、ドライブ A のルートディレクトリから順に検索していく。

ある程度検索したところでダイアログを表示する。
dNamePtr は sNamePtr と同じものを指定してもよい。
再配置が発生する。

Cの関数► int *TSSearchFile*(const char *sNamePtr, char *dNamePtr, const char *path);
返り値は、ファイルサイズまたはリザルトコード。

$A406　SXStrCmp

引　数►

long	sStrPtr	; 文字列 1 へのポインタ
long	dStrPtr	; 文字列 2 へのポインタ
word	length	; 比較するバイト数

返り値► D0.L　= 0　一致
　　　　　$\neq$ 0　一致しない

機　能► sStrPtr で指定された文字列と dStrPtr で指定された文字列を、length バイトだけ大文字・小文字を無視して比較する。

Cの関数► int *SXStrCmp*(const char *sStrPtr, const char *dStrPtr, int length);
返り値は結果を意味する数値。

$A408　TSCreateISBadge

引　数►

long	ISRecPtr	; アイコン管理レコードへのポインタ
word	mediabyte	; メディアバイト
word	unitno	; ユニット番号

返り値► D0.L　リザルトコード

機　能► mediabyte, unitno で指定したメディアバイトとユニット番号からアイコン管理レコードを作成する。
再配置が発生する。

Cの関数► int *TSCreateISBadge*(IcState *ISRecPtr, int mediabyte, int unitno);
返り値はリザルトコード。

$A40A　TSGetCMDS

引　数►

word	cmdID	; ビルトインコマンドの番号
long	buffPtr	; コマンド名 (ASCIIZ 型) が返るバッファのアドレス

返り値► D0.L　リソース 'CODE' の ID/リザルトコード

A0.L　　コマンド名の終端 + 1

機　能► cmdID で指定したビルトインコマンドの名前 (ASCIIZ 型) と、コマンドのコードが収められているリソース 'CODE' の ID を返す。
再配置が発生する。

Cの関数► int *TSGetCMDS*(int cmdID, char *buffPtr);
返り値は、リソース ID またはリザルトコード。

$A40B　　TSFockCM

引　数►

byte	mode1	; 起動モード 1
byte	mode2	; 起動モード 2
long	cmdsID	; リソース "CMDS" の ID
long	commandPtr	; コマンドライン文字列 (LASCII 型) へのポインタ
long	environPtr	; 環境へのポインタ
long	dFilename	; 実際に起動したファイル名が返るバッファ (90 バイト) のアドレス

返り値► D0.L　　タスク ID/リザルトコード

機　能► cmdsID で指定した番号のビルトインコマンドを起動する。
タスクには、commandPtr で指定したコマンドライン文字列と environPtr で指定した環境が渡される。environPtr として 0 を指定すると、タスクマンの環境を使用することになる。
mode1 は $A351 TSFock と同様の意味を持つ。
mode2 として 0 以外を指定した場合、起動した (しようとした) ファイル名を dFilename に返す。
再配置が発生する。

Cの関数► int *TSFockCM*(int mode1,BOOLEAN mode2,int cmdsID,const char *commandPtr,const char *environPtr,char *dFilename);
返り値は、タスク ID またはリザルトコード。

$A40D　　TSTiniTsk

引　数► なし

返り値► D0.L　　リザルトコード

機　能► タスクマンの終了処理 (システムリソースのクローズ、すべてのマネージャの終了処理) を行う。
再配置が発生する。

Cの関数► int *TSTiniTsk*(void);
返り値はリザルトコード。

$A415 TSPostEventTsk2

引　数► long mes1 ；メッセージ1
long mes2 ；メッセージ2
word what2 ；タスクマンイベントコード
byte Hmode1 ；ハンドルモード1
byte Hmode2 ；ハンドルモード2
word taskID ；イベントを発生させるタスクのID

返り値► D0.L リザルトコード

機　能► mes1, mes2, what2で指定したデータを、それぞれwhom, whom2, what2に持つようなタスクマンイベントレコードを作成し、taskIDで指定したタスクにイベントとして発生させる。このイベントの種類は12(E_SYSTEM1)に固定。

whomあるいはwhom2に格納されている引数がハンドルである場合、Hmode1あるいはHmode2に 0 以外を指定すると、別のブロックを作成してwhom, whom2が指すブロックの内容をコピーする。タスクマンのイベントキューに登録されるタスクマンイベントレコードのwhom, whom2には新しく作成したブロックへのハンドルが格納される。この場合、タスクマンイベントレコードを廃棄する際、これらのブロックは同時に廃棄される。

再配置が発生する。

Cの関数► int *TSPostEventTsk2*(long mes1, long mes2, int what2, BOOLEAN Hmode1, BOOLEAN Hmode2, int taskID);

返り値はリザルトコード。

$A417 TSAnswer

引　数► long tsEventRecPtr ；タスクマンイベントレコードへのポインタ

返り値► D0.L

0	正常終了
-1	通信中ではない
-2	宛先のタスクの準備が整っていない

機　能► tsEventRecPtrで指定したイベントレコードの内容を、タスク間通信に対する返事として通信相手のタスクに返す。

再配置が発生する。

Cの関数► int *TSAnswer*(TsEvent *tsEventRecPtr);

返り値は結果を意味する数値。

$A418 TSSendMes

引　数► word listener ；宛先となるタスクのID
long tsEventRecPtr ；タスクマンイベントレコードへのポインタ

返り値► D0.L

0	正常終了
2	返事が届いた
-1	宛先のタスクが存在しない/通信中
-2	宛先のタスクの準備が整っていない

機　能► listener で指定したタスクに tsEventRecPtr で指定したタスクマンイベントレコードの内容をメッセージとして送信する。返事があった場合は、tsEvent RecPtr のなかに返事が返る。
再配置が発生する。

Cの関数► int *TSSendMes*(int listener, TsEvent *tsEventRecPtr);
返り値は結果を意味する数値。

$A419　　TSGetMes

引　数► long　tsEventRecPtr　；タスクマンイベントレコードへのポインタ
word　mode

0	受け付けなかったことにする
≠0	受け付けたことを通知する

返り値► D0.L

0	メッセージは届いていない
1	メッセージを受け取った

機　能► tsEventRecPtr で指定したタスクマンイベントレコードにメッセージを読み込む。mode で 0 を指定した場合は、相手には -2（「宛先のタスクの準備が整っていない」）が、0 以外を指定した場合は 0（「正常終了」）が返る。
起動直後にメッセージを受け付ける場合に使用する。

Cの関数► int *TSGetMes*(TsEvent *tsEventRecPtr, BOOLEAN mode);
返り値は、結果を意味する数値。

$A41A　　TSInitTsk2

引　数► long　memStart　；ヒープゾーン先頭アドレス
long　memEnd　；ヒープゾーン終了アドレス
long　pathPtr　；カレントパスを示す文字列（ASCIIZ 型）へのポインタ
byte　mode1　；シェルのリリース番号（1 ～ ）
byte　mode2　；システムリソースヒープゾーン作成フラグ

0	作成しない
≠ 0	作成する

word　ver　；シェルのバージョン
long　rscfilePtr　；システムリソース名（ASCIIZ 型）へのポインタ（90 バイト）

返り値► D0.L　ヒープゾーンのアドレス/リザルトコード
A0.L　システムリソースへのハンドル

機　能► タスクマン以下のマネージャをすべて初期化する。
$A34C TSInitTsk との違いは、rscfilePtr によってシステムリソースファイルを指定できる点。rscfilePtr が 0、あるいは先頭 1 バイトが 0 の場合、デフォルトの "SYSTEM.LB" が使用される。また、実際にオープンしたファイル名が rscfilePtr に格納されるため、文字列を格納する領域は 90 バイト必要。

Cの関数► Heap *TSInitTsk2(void *memStart, void *memEnd, const char *pathPtr,int mode1,BOOLEAN mode2,int ver,char *rscfilePtr, Handle *resHdl);
返り値は、ヒープゾーンのアドレスまたはリザルトコード。
void 型のハンドル resHdl にはシステムリソースへのハンドルが格納される。

$A41F　SXCallWindM2

引　数► long　winPtr　; ウィンドウレコードへのポインタ
long　tsEventRecPtr　; タスクマンイベントレコードへのポインタ
long　rectPtr　; レクタングルレコードへのポインタ

返り値► D0.L　ウィンドウのパートコード/リザルトコード

機　能► $A3A2 SXCallWindM と同様の処理を行う。
rectPtr で指定するレクタングルでウィンドウサイズ変更時の最大サイズ、最小サイズを指定できる点が異なる。左上の座標が最小サイズ、右下の座標が最大サイズを意味する。
再配置が発生する。

Cの関数► int SXCallWindM2(Window *winPtr,TsEvent *tsEventRecPtr, Rect *rectPtr);
返り値は、ウィンドウのパートコードまたはリザルトコード。

$A420　TSBeginDrag2

引　数► long　pt　; ドラッグ開始位置（グローバル座標）
long　procPtr　; ラバーバンド表示ルーチンのアドレス
long　param　; ラバーバンド表示ルーチンに渡されるパラメータ

返り値► D0.L　リザルトコード

機　能► pt で指定したポイントからドラッグを開始する。$A38A TSBeginDrag とは、ラバーバンド表示ルーチンを指定できる点で異なる。procPtr で 0 を指定すると標準のラバーバンド表示ルーチンが使用されるが、この場合はアイコン管理レコードのみサポートされる。
あらかじめカレントグラフレコードをセットしておく必要がある。
ラバーバンド表示ルーチンへは、次のようなパラメータが A0 経由で渡される。

(a0).L　ドラッグレコードへのポインタ
4(a0).W　コマンド
6(a0).L　TSBeginDrag2 で指定したパラメータ

コマンドには「初期化」、「表示」、「消去」、「終了」の4つが存在し、その仕様は以下のとおり。

- command = 0 : 初期化
  最初に1度だけ呼ばれる。標準のラバーバンド表示ルーチンではビットイメージの作成が行われる。
- command = 1 : 終了
  終了時に呼ばれる。初期化時に確保したメモリの廃棄等を行う。
- command = 2 : 表示
  ラバーバンドを表示する。ドラッグレコードの drOrigin を引数にして $A13B GMSetHome をコールすることで移動した分のローカル座標がセットされる。
- command = 3 : 消去
  ラバーバンドを消去する。ドラッグレコードの drOrigin を引数にして $A13B GMSetHome をコールすることで移動した分のローカル座標がセットされる。

Cの関数► int *TSBeginDrag2*(LPoint pt, void *procPtr, long param);
返り値はリザルトコード。

$A422　SXGetVector

引　数► word　intNo　;SX コール番号
返り値► D0.L　ベクタの内容/リザルトコード
A0.L　ベクタの内容
機　能► intNo で指定した SX コールのベクタを返す。
Cの関数► void **SXGetVector*(int intNo);
返り値は、ベクタの内容またはリザルトコード。

$A423　SXSetVector

引　数► word　intNo　;SX コール番号
long　vector　;登録するアドレス
返り値► D0.L　前のベクタの内容/リザルトコード
A0.L　前のベクタの内容
機　能► intNo で指定した SX コールのベクタとして vector を設定する。
Cの関数► void **SXSetVector*(int intNo, void *vector);
返り値は、前のベクタの内容またはリザルトコード。

タスクマン

$A427 TSCellToStr

引　数► long celHdl ；セルリストへのハンドル
long buffPtr ；結果が返るバッファのアドレス
long max ；バッファのサイズ

返り値► D0.L 文字列のバイト数/リザルトコード
A0.L 文字列を収めたバッファへのハンドル（確保しなかった場合は 0）

機　能► celHdl で指定したセルリストから文字列を抽出し、buffPtr で指定したバッファに返す。max で指定したバッファのサイズを超えた場合、エラーとなる。
buffPtr として 0 を指定した場合、ヒープ上に再配置可能ブロックを作成し、そこに納めてハンドルを A0 レジスタに返す。
再配置が発生する。

Cの関数► long *TSCellToStr*(Handle celHdl, char *buffPtr, long max, char ***buffHdl);
返り値は、文字列のバイト数またはリザルトコード。
char 型のハンドル buffHdl には、文字列を納めたバッファへのハンドルが格納される。

$A42A SXLockFSX

引　数► なし

返り値► なし

機　能► SX システムをロックする。

Cの関数► void *SXLockFSX*(void);
返り値はない。

$A42B SXUnlockFSX

引　数► なし

返り値► なし

機　能► SX システムをアンロックする。

Cの関数► void *SXUnlockFSX*(void);
返り値はない。

$A42C TSFockMode

引　数► long fileNamePtr ；ファイル名（ASCIIZ 型）へのポインタ
long dFileName ；実際に起動されるファイル名が返るバッファ（90 バイト）

返り値► D0.L

0	ファイルから起動
1	リソースタイプ'CODE'から起動
2	メモリ中のタスクを複写して起動

A0.L　複写するタスクのID/リソース'CODE'のID

機　能► fileNamePtrで指定したファイルを $A353 TSFockBで起動した場合の起動モードを調べる(起動するわけではない)。dFileNameとfileNamePtrは同じものを指定してもよい。
再配置が発生する。

Cの関数► int *TSFockMode*(const char *fileNamePtr, char *dFileName, long *taskID);
返り値は、結果を意味する数値。
int 型の変数 taskID には、複写するタスクのIDまたはリソース'CODE'のIDが格納される。

$A430　TSSetGraphMode

引　数► word　rmode　;0 以外で実画面モード
word　scrMode　;画面モード

返り値► D0.L　リザルトコード

機　能► タスクマンを初期化する際に参照される画面モードをSRAMに設定する。scrModeは、IOCSの _CRTMOD で指定する値と同等。rmode として 0 以外を指定すると、実画面モードになる。

Cの関数► int *TSSetGraphMode*(BOOLEAN rmode, int scrMode);
返り値はリザルトコード。

$A431　TSGetGraphMode

引　数► なし

返り値► D0.L　SRAMに設定されている画面モード
上位ワード：　0 以外で実画面モード
下位ワード：　画面モード
A0.L　グラフマンを初期化した値
上位ワード：　0 以外で実画面モード
下位ワード：　画面モード
= -1 グラフマンは初期化されていない

機　能► タスクマンの画面モードを得る。

Cの関数► long *TSGetGraphMode*(long *initValue);
返り値はSRAMに設定されている画面モード。
long 型の変数 initValue にはグラフマンを初期化した値が格納される。

$A432　SXGetDispRect

引　数► long　rectPtr　；レクタングルレコードへのポインタ
返り値► なし
機　能► rectPtr で指定したレクタングルレコードに現在の表示画面を返す。実画面モードの場合、表示画面と実際に描画を行う実画面は異なる。
Cの関数► int *SXGetDispRect*(Rect *rectPtr);
返り値は意味を持たない。

$A435　SXSRAMVer

引　数► なし
返り値► D0.L　バージョン

0	SX-WINDOW 1.10 以前
1	SX-WINDOW 1.10 以降
2	SX-WINDOW 3.00 以降

機　能► SRAM を初期化した SX システムのバージョンを得る。
Cの関数► int *SXSRAMVer*(void);
返り値は、SRAM を初期化した SX システムのバージョン。

$A436　SXSRAMReset

引　数► なし
返り値► なし
機　能► SX システムが使用する SRAM 領域を初期化する。
Cの関数► void *SXSRAMReset*(void);
返り値はない。

$A437　SXSRAMCheck

引　数► なし
返り値► D0.L

0	自分と同じバージョンなので初期化を行わなかった
1	初期化を行った
2	自分より新しいバージョンなので初期化を行わなかった

機　能► SRAM に記録されている情報のバージョンを調べ、自分より古い場合は初期化を行う。
Cの関数► int *SXSRAMCheck*(void);
返り値は、結果を意味する数値。

$A438　TSAdjustRect　2.0

引　数► long　dRectPtr　; 結果を格納するレクタングルレコードへのポインタ
long　sRectPtr　; もとになるレクタングルレコードへのポインタ
word　mode　; 画面モード

返り値► なし

機　能► mode の画面モードで sRectPtr を表示した場合の画面の位置を、現在の画面モードで表示した場合に換算して dRectPtr に格納する。
画面モードやスクロールレジスタの状態にかかわらず、固定した位置にウィンドウをオープンする場合などに使用する。
dRectPtr と sRectPtr は同じものが指定できる。
mode の値は $A430 TSSetGraphMode と同様。

Cの関数► void *TSAdjustRect*(Rect *dRectPtr, Rext *sRectPtr, int mode);
返り値はない。

$A43B　TSPostEventTsk3　2.0

引　数► long　mes1　;メッセージ1
long　mes2　;メッセージ2
word　what2　;タスクマンイベントコード
byte　Hmode1　;ハンドルモード1

0	そのまま使用(廃棄しない)
1	そのまま使用(廃棄する)
2	コピーして使用(廃棄しない)
3	コピーして使用(廃棄する)

byte　Hmode2　;ハンドルモード2

0	そのまま使用(廃棄しない)
1	そのまま使用(廃棄する)
2	コピーして使用(廃棄しない)
3	コピーして使用(廃棄する)

word　taskID　;イベントを発生させるタスクのID
long　size1　;mes1 のサイズ
long　size2　;mes2 のサイズ

返り値► D0.L　リザルトコード

機　能► mes1, mes2, what2 で指定したデータをそれぞれ whom, whom2, what2 に持つようなタスクマンイベントレコードを作成し、taskID で指定したタスクにイベントとして発生させる。このイベントの種類は 12(E_SYSTEM1) に固定。

whom あるいは whom2 に格納されている引数がハンドルである場合、Hmode1 あるいは Hmode2 の bit1 を 1 にする、別のブロックを作成して whom, whom2 が指すブロックの内容をコピーする。タスクマンのイベントキューに登録されるタスクマンイベントレコードの whom, whom2 には新しく作成したブロックへのハンドルが格納される。bit0 を 1 にした場合、タスクマンイベントレコードを廃棄する際、これらのブロックは同時に廃棄される。
taskID に -1 を指定した場合、すべてのタスクにイベントが発生する。
size1(または size2) に -1 を指定した場合、mes1(または mes2) のブロックのサイズが用いられる。この場合、mes1(または mes2) は疑似ハンドルであってはならない。
再配置が発生する。

Cの関数► void *TSPostEventTsk3*(long mes1, long mes2, int what2, int Hmode1, int Hmode2, int taskID, long size1, long size2);
返り値はない。

$A43E TSAnswer2 2.0

引 数►

long	mes1	; メッセージ 1
long	mes2	; メッセージ 2
word	what2	; タスクマンイベントコード
byte	Hmode1	; ハンドルモード 1

0	そのまま使用 (廃棄しない)
1	そのまま使用 (廃棄する)
2	コピーして使用 (廃棄しない)
3	コピーして使用 (廃棄する)

byte	Hmode2	; ハンドルモード 2

0	そのまま使用 (廃棄しない)
1	そのまま使用 (廃棄する)
2	コピーして使用 (廃棄しない)
3	コピーして使用 (廃棄する)

word	taskID	; イベントを発生させるタスクの ID
long	size1	;mes1 のサイズ
long	size2	;mes2 のサイズ

返り値► D0.L

0	$A417 TSAnswer で返事を返した
1	タスク指定イベントを登録した
< 0	リザルトコード

機 能► 指定した引数によってタスクマンイベントレコードを作成し、タスク間通信に対する返事を返す。$A417 TSAnswer でエラーが発生した場合はタスク指定イベントを登録する。
各引数の内容については $A43B TSPostEventTsk3 を参照。
タスクの切り替えは発生しない。

再配置が発生する。

Cの関数► int *TSAnswer2*(long mes1,long mes2,int what2,int Hmode1, int Hmode2, int taskID, long size1, long size2);
返り値は、結果を意味する数値またはリザルトコード。

$A443 TSErrDialogN 2.0

引　数► word flags ; ダイアログの形式を指定するフラグ
long strZPtr ; エラー/警告メッセージ (ASCIIZ 型) へのポインタ
返り値► D0.L アイテム番号/リザルトコード
機　能► タスク名表示付きの簡易エラーメッセージ用ダイアログを表示する。タスク名が表示される以外は $A2F6 DMError と同様。
再配置が発生する。
Cの関数► int *TSErrDialogN*(int flags, const char *strZPtr);
返り値は、アイテム番号またはリザルトコード。

$A446 TSSearchFile3 2.0

引　数► byte mode1 ; ダイアログを表示するまでの検索回数
byte mode2 ; 検索条件

0	指定しない
1	path で指定されたパスリストと sNamePtr で指定されたドライブだけを検索する
2	path で指定されたパスリストだけを検索する

long sNamePtr ; ファイル名 (ASCIIZ 型) へのポインタ
long dNamePtr ; 発見したファイル名を格納するバッファ(90 バイト)
long path ; パスリスト文字列 (ASCIIZ 型) へのポインタ
long filterProc ; フィルタプロセスのアドレス
返り値► D0.L ファイルサイズ/リザルトコード
機　能► sNamePtr で指定したファイルを mode1, mode2 に従って検索し、発見できたファイルの名前を dNamePtr に返す。sNamePtr のなかにパスが含まれない場合、カレントパスが補われる。
path でポインタを指定するパスリストは、1 個以上のパス名を“;”で区切って並べた ASCIIZ 文字列。path として 0 を指定すると、検索パスを指定しないことになる。
filterProc は、発見したファイルを取捨選択するための関数のアドレス。関

数の仕様は以下のとおり。

```
int (*filterProc)(const char *name, long ext);
```

name は、パスを除いたファイル名へのポインタ。
ext は拡張子。
返り値として負の値を返すと、そのファイルは発見しなかったことになる。

dNamePtr は sNamePtr と同じものを指定してもよい。
再配置が発生する。

Cの関数► long *TSSearchFile3*(BOOLEAN mode1, int mode2, const char sNemePtr, char *dNamePtr, const char *path, int (*filterProc) (const char *name, long ext));
返り値は、ファイルサイズまたはリザルトコード。

$A44B TSNameToCode 3.0

引　数► long eventNamePtr ；イベント名（ASCIIZ 型）へのポインタ
返り値► D0.L イベントコード（$9000 ～ $BFFF）/リザルトコード
機　能► eventNamePtr で指定した名前のイベントコードを得る。同名のイベントが未登録の場合は登録する。
再配置が発生する。

Cの関数► int *TSNameToCode*(char *eventNamePtr);
返り値は、イベントコードまたはリザルトコード。

$A44C TSCodeToName 3.0

引　数► word eventCode ；イベントコード
long buffPtr ；イベント名（ASCIIZ 型）が返るバッファへのポインタ
返り値► D0.L リザルトコード
機　能► eventCode で指定したイベントコードに対応するイベント名を buffPtr で示されたバッファに格納する。
再配置が発生する。

Cの関数► int *TSCodeToName*(int eventCode, char *buffPtr);
返り値はリザルトコード。

$A44D　TSNameToHdl　3.0

引　数► long　strZPtr　;ASCIIZ 文字列へのポインタ
返り値► D0.L　再配置可能ブロックへのハンドル

0	エラー
< 0	リザルトコード

機　能► strZPtr で指定した文字列に対応するハンドルを返す。同名のハンドルが登録されていなかった場合は 0 バイトの再配置可能ブロックを作成し、登録する。
再配置が発生する。

Cの関数► Handle *TSNameToHdl*(char *strZPtr);
返り値はハンドルまたはリザルトコード。
エラーが発生した場合、NULL が返る。

$A450　SXPack　3.1

引　数► word　pID　; データ圧縮定義関数の ID
long　srcLen　; 圧縮前のデータのバイト数
long　destPtr　; 圧縮後のデータを格納するバッファへのポインタ
long　srcPtr　; 圧縮前のデータへのポインタ
返り値► D0.L　圧縮後のデータのサイズ/リザルトコード
A0.L　destPtr として 0 を指定した場合、データを格納したハンドル
機　能► srcPtr で指定した srcLen バイトのデータを、pID で指定したデータ圧縮定義関数によって圧縮する。
結果は destPtr で指定したバッファに格納される。destPtr として 0 を指定した場合、データ圧縮定義関数が再配置可能ブロックを確保し、そのなかで圧縮後のデータを格納する。このハンドルは A0.L に返される。
再配置が発生する。

Cの関数► int *SXPack*(short pID, long srcLen, void *destPtr, void *srcPtr, Handle *destHdl);
destPtr が NULL の場合、圧縮データのハンドルはハンドル destHdl に格納される。
返り値は圧縮後のデータのサイズまたはリザルトコード。

$A451　SXUnpack　3.1

引　数► word　pID　; データ圧縮定義関数の ID
long　srcLen　; 展開前のデータのバイト数

long　destPtr　; 展開後のデータを格納するバッファへのポインタ
long　srcPtr　; 展開前のデータへのポインタ

返り値► D0.L　展開後のデータのサイズ/リザルトコード
A0.L　destPtr として 0 を指定した場合、データを格納したハンドル

機　能► srcPtr で指定した srcLen バイトのデータを、pID で指定したデータ圧縮定義関数によって展開する。
結果は、destPtr で指定したバッファに格納される。destPtr として 0 を指定した場合、データ圧縮定義関数が再配置可能ブロックを確保し、そのなかで展開後のデータを格納する。このハンドルは A0.L に返される。
再配置が発生する。

Cの関数► int *SXUnpack*(short pID, long srcLen, void *destPtr, void *srcPtr, Handle *destHdl);
destPtr が NULL の場合、展開データのハンドルはハンドル destHdl に格納される。
返り値は展開後のデータのサイズまたはリザルトコード。

$A452　SXGetPackSize　3.1

引　数► word　pID　; データ圧縮定義関数の ID
long　srcLen　; 圧縮前のデータのバイト数

返り値► D0.L　圧縮後に予想される最大のサイズ/リザルトコード

機　能► srcLen のサイズのデータを pID で指定するデータ圧縮定義関数で圧縮した場合、最悪のケースでどれだけのサイズのバッファが必要かを返す。
再配置が発生する。

Cの関数► int *SXGetPackSize*(short pID, long srcLen);
返り値は、圧縮後に予想される最大サイズまたはリザルトコード。

$A453　SXGetCODFList　3.1

引　数► なし

返り値► D0.L　データ圧縮定義関数の数
A0.L　配列へのハンドル

機　能► システムに登録されているデータ圧縮定義関数に関する情報を、配列の形で再配置可能ブロックに入れて返す。
配列の 1 つのレコードは、以下のような構造を持つ。

```
typedef struct {
      short      resID;            /*−1 のとき終端 */
      long       kind;             /*セルデータの種類を意味する識別子 */
      unsigned char name[22];      /* ASCIIZ (名前がない場合もある) */
      long       rsv;
} CODFInfo;
```

データ圧縮定義関数は、システムリソースにリソースタイプ'CODF'のリソースとして用意される。'CODF'リソースの内容は単なる関数で、以下のような仕様を備えている。

```
int compactDef(short cmd, long *argv);
cmd      :  コマンド (0:圧縮、1:展開、2:圧縮後の最大サイズ計算)
argv[0]  :  もとになるデータのバイト数 (cmd = 0, 1, 2)
argv[1]  :  処理済みデータを書き出すバッファへのポインタ (cmd = 0, 1)
argv[2]  :  処理を行うデータへのポインタ (cmd = 0, 1)
```

返り値はリザルトコード。

データ圧縮定義関数は、'CODF'と同じIDを持つ情報リソース'COIF'とペアとなることでシステムに認知される。'COIF'リソースは以下の構造を持つ。

```
typedef struct {
    long          kind;          /* セルデータの種類を意味する識別子*/
    unsigned char name[22];      /* ASCIIZ (名前がない場合もある) */
} COIF;
```

SX WINDOW ver.3.1 は、SX システム内にデフォルトのデータ圧縮定義関数を持つ (リソースでオーバーライドすることも可)。

ID	kind	圧縮アルゴリズム
0	CORL	ランレングス圧縮
1	CORP	ランレングスピクセル圧縮

再配置が発生する。

Cの関数► int *SXGetCODFList*(CODFInfo ***ciHdl);

データ圧縮定義関数に関する情報の格納されている再配置可能ブロックのハンドルが、CODFInfo 型のハンドル ciHdl に格納される。

返り値はデータ圧縮定義関数の数。

$A454　SXCellToCODF　3.1

引　数► long　kind　；セルデータの種類を意味する識別子

返り値► D0.L　データ圧縮定義関数のID/リザルトコード

機　能► kind で指定されたセルデータの種類から、そのセルデータを展開可能なデータ圧縮定義関数のIDを返す。

セルリストのなかのセルの種類が'CO??'だった場合、アプリケーションは

このコールを使って展開可能なデータ圧縮定義関数を探すことができる。
圧縮されたセルは、以下のような構造を持つ。

```
typedef struct {
    unsigned long  kind;     /* 'CO??'  */
    long           length;   /* 圧縮後のサイズ */
    unsigned long  oKind;    /* 圧縮前のセルの種類 */
    long           oLength;  /* 圧縮前のサイズ */
    unsigned char  data[2];  /* 圧縮されたデータ */
} cCell;
```

このような形式のセルを$A451 SXUnpack で展開する場合、展開前のデータへのポインタとして data の位置を指定する。
再配置が発生する。

Cの関数► int *SXCellToCODF*(long kind);
返り値はデータ圧縮定義関数の ID またはリザルトコード。

$A4A0　TSResNew　2.0

引　数► なし

返り値► D0.L　リザルトコード
A0.L　リソースマップへのハンドル

機　能► リソースマップをメモリ上に作成し、コールしたタスクのリソース系列のカレントとする。リソース系列がリンクされていない場合は、新しいローカルリソース系列を割り当てる。実際にファイルが作成されるわけではない。
作成直後のリソースマップの内容は空である。
再配置が発生する。

Cの関数► Handle *TSResNew*(void);
返り値はリソースマップへのハンドル。

$A4A1　TSResOpen　2.0

引　数► long　Name　；リソースファイル名

返り値► D0.L　リンクしたリソース系列/リザルトコード
A0.L　リソースマップへのハンドル

機　能► Name で指定したリソースファイルをオープンして、コールしたタスクのリソース系列のカレントとする。リソース系列がリンクされていない場合は、新しいローカルリソース系列を割り当てる。この時点ではリソースマップのみが読み込まれ、ファイルはオープン状態のままである。
再配置が発生する。

Cの関数► Handle *TSResOpen*(const char *Name);

返り値はリソースマップへのハンドル。エラーが発生した場合、NULL が返る。

$A4A2　TSResClose　2.0

引　数► long　Name　; ファイル名

返り値► D0.L　リザルトコード

機　能► コールしたタスクのリソース系列のカレントリソースマップと、その下のリソース群を Name で指定したファイルにセーブして、メモリから削除する。カレントは次のリソースマップに移る。

再配置が発生する。

Cの関数► int *TSResClose*(const char *Name);

返り値はリザルトコード。

$A4A3　TSResSave　2.0

引　数► long　Name　; リソースファイル名

返り値► D0.L　リザルトコード

A0.L　リソースファイル名がそのまま返る

機　能► コールしたタスクのリソース系列のカレントリソースマップと、その下のリソース群を Name で指定したファイルにセーブする。メモリから削除しない。カレントリソースマップは変更されない。

再配置が発生する。

Cの関数► int *TSResSave*(const char *Name);

返り値はリザルトコード。

$A4A4　TSResRemove　2.0

引　数► なし

返り値► D0.L　リザルトコード

機　能► コールしたタスクのリソース系列のカレントリソースマップと、その下のリソース群をメモリから削除する。オープン中の場合はクローズする。カレントは次のリソースマップに移る。

再配置が発生する。

Cの関数► int *TSResRemove*(void);

返り値はリザルトコード。

$A4A5　TSResLoad　2.0

引　数► なし

返り値► D0.L　　リザルトコード

機　能► コールしたタスクのリソース系列のカレントリソースマップと以下のリソースを、すべて読み込んでメモリに置く。ファイルはクローズされる。
再配置が発生する。

Cの関数► int *TSResLoad*(void);
返り値はリザルトコード。

$A4A6　　TSResDispose　　2.0

引　数► なし

返り値► D0.L　　リザルトコード

機　能► コールしたタスクのリソース系列のカレントリソースマップをメモリから削除する。オープン中の場合でもクローズしない。
再配置が発生する。

Cの関数► int *TSResDispose*(void);
返り値はリザルトコード。

$A4A7　　TSCurResGet　　2.0

引　数► なし

返り値► D0.L　　カレントリソースマップへのハンドル
A0.L　　カレントリソースマップへのハンドル

機　能► コールしたタスクのリソース系列のカレントリソースマップへのハンドルを返す。

Cの関数► Handle *TSCurResGet*(void);
返り値はカレントリソースマップへのハンドル。

$A4A8　　TSLastResGet　　2.0

引　数► なし

返り値► D0.L　　最終リソースマップへのハンドル
A0.L　　最終リソースマップへのハンドル

機　能► コールしたタスクの、リソース系列の最終リソースマップ（最後にオープンされたリソースマップ）へのハンドルを返す。

Cの関数► Handle *TSLastResGet*(void);
返り値は、最終リソースマップへのハンドル。

$A4A9　　TSCurResSet　　2.0

引　数► long　　ResMap　　　　; リソースマップへのハンドル

返り値► D0.L　リザルトコード
A0.L　前のカレントリソースマップへのハンドル

機　能► ResMap で指定したメモリ中のリソースマップを、コールしたタスクのリソース系列のカレントにする。
リソース系列にリンクされていないリソースマップを指定した場合はエラーになる。
リソース系列を共有するすべてのタスクで、カレントリソースマップが変更される。

Cの関数► Handle *TSCurResSet*(Handle ResMap);
返り値は、前のカレントリソースマップへのハンドル。

$A4AA　TSRscAdd　2.0

引　数► long　Type　;追加するリソースのタイプ
word　ID　;追加するリソース ID
long　h　;追加するリソースへのハンドル
long　Size　;追加するリソースのサイズ

返り値► D0.L　リザルトコード
A0.L　新しいリソースへのハンドル

機　能► コールしたタスクの、リソース系列のカレントのリソースマップにリソースを追加する。
リソースは、リソースマンが新しく確保したブロックにコピーされる。疑似ハンドルも可。
再配置が発生する。

Cの関数► _Handle *TSRscAdd*(long Type,int ID,_Handle h,long Size);
返り値は新しいリソースへのハンドル。

$A4AB　TSRscGet　2.0

引　数► long　Type　;Get したいリソースのタイプ
word　ID　;Get したいリソース ID

返り値► D0.L　リザルトコード
A0.L　リソースへのハンドル

機　能► コールしたタスクのリソース系列から Type と ID で指定したリソースを探し、発見できた場合はハンドルを返す。
再配置が発生する。

Cの関数► _Handle *TSRscGet*(long Type, int ID);
返り値はリソースへのハンドル。エラーが発生した場合は NULL が返る。

$A4AC　TSRscRemove　2.0

引　数► long　Type　;削除するリソースのタイプ
word　ID　;削除するリソース ID
返り値► D0.L　リザルトコード
機　能► Type と ID で指定したリソースを、コールしたタスクのリソース系列のカレントリソースマップから削除する。
再配置が発生する。
Cの関数► int *TSRscRemove*(long Type, int ID);
返り値はリザルトコード。

$A4AD　TSTypeRemove　2.0

引　数► long　Type　;削除するリソース群のタイプ
返り値► D0.L　リザルトコード
機　能► コールしたタスクのリソース系列のカレントリソースマップの、Type で指定するタイプのリソース群をすべて削除する。
再配置が発生する。
Cの関数► int *TSTypeRemove*(long Type);
返り値はリザルトコード。

$A4AE　TSRscRelease　2.0

引　数► long　RscHdl　;Release するリソースのハンドル
返り値► D0.L　リザルトコード
A0.L　ハンドルがそのまま返る（エラーの場合は 0）
機　能► RscHdl で指定したリソースを、タスクマン（ローカルリソースマネージャ）の管理から外し、メモリから削除する。
コールしたタスクのリソース系列にリンクされていないリソースを指定した場合はエラーになる。
再配置が発生する。
Cの関数► int *TSRscRelease*(_Handle RscHdl);
返り値はリザルトコード。

$A4AF　TSRscDetach　2.0

引　数► long　RscHdl　;Detach するリソースのハンドル
返り値► D0.L　リザルトコード
A0.L　ハンドルがそのまま返る（エラーの場合は 0）

機　能► RscHdlで指定したリソースを、タスクマン（ローカルリソースマネージャ）の管理から外す。メモリからは削除しない。
コールしたタスクのリソース系列にリンクされていないリソースを指定した場合はエラーになる。
再配置が発生する。

Cの関数► _Handle *TSRscDetach*(_Handle RscHdl);
返り値は、タスクマンの管理から外したリソースへのハンドル。

$A4B0　TSMaxIDGet　2.0

引　数► long　Type　;最大IDを調べるタイプ

返り値► D0.L　Type の最大ID/リザルトコード

機　能► Type で指定したリソース群のなかで、コールしたタスクのリソース系列中で最大のIDを返す。

Cの関数► int *TSMaxIDGet*(long Type);
返り値は、 Type の最大IDまたはリザルトコード。

$A4B1　TSHdlToRsc　2.0

引　数► long　RscHdl　;リソースマップを得るリソースへのハンドル

返り値► D0.L　リソースマップへのハンドル

機　能► RscHdl で指定した、リソースが所属するリソースマップのハンドルが返る。
コールしたタスクのリソース系列にリンクされていないリソースを指定した場合はエラーになる。

Cの関数► Handle *TSHdlToRsc*(_Handle RscHdl);
返り値はリソースマップへのハンドル。

$A4B2　TSResLinkGet　2.0

引　数► long　ResMapHdl　;リソースマップへのハンドル

返り値► D0.L　リソースマップへのハンドル

機　能► ResMapHdl で指定したリソースマップの次のリソースマップを返す。

Cの関数► Handle *TSResLinkGet*(Handle ResMapHdl);
返り値はリソースマップへのハンドル。

$A4B3　TSResRouteLink　2.0

引　数► word　taskID　;タスクID
long　route　;リンクするリソース系列

返り値► D0.L　　リンクカウント/リザルトコード
A0.L　　前のリソース系列

機　能► taskID で指定したタスクに route で指定したリソース系列をリンクする。
taskID として -1 を指定すると、コールしたタスクにリンクする。

Cの関数► long *TSResRouteLink*(int taskID, long route);
返り値は前のリソース系列。

$A4B4　TSResRouteGet　2.0

引　数► word　taskID　;タスク ID

返り値► D0.L　リソース系列

機　能► taskID で指定したタスクのリソース系列を返す。
taskID として -1 を指定すると、コールしたタスクのリソース系列を返す。

Cの関数► long *TSResRouteGet*(int taskID);
返り値はリソース系列。

$A4B5　TSRscGet2　2.0

引　数► long　Type　;Get したいリソースのタイプ
word　ID　;Get したいリソース ID
long　route　;リソース系列

返り値► D0.L　リザルトコード
A0.L　リソースへのハンドル

機　能► route で指定したリソース系列から Type と ID で指定したリソースを探し、発見できた場合はハンドルを返す。
再配置が発生する。

Cの関数► _Handle *TSRscGet2*(long Type, int ID, long route);
返り値はリソースへのハンドル。エラーが発生した場合は NULL が返る。

$A4B6　TSRscGet3　2.0

引　数► long　Type　;Get したいリソースのタイプ
word　ID　;Get したいリソース ID
long　route　;リソース系列

返り値► D0.L　リザルトコード
A0.L　リソースへのハンドル

機　能► コールしたタスクの系列、システムリソースの順に検索し、Type と ID で指定したリソースが発見できた場合はハンドルを返す。
SX-WINDOW ver.3.0 以降では、タスクマンが$A0E1 RMRscGet をフックし

て、このコールと同様の働きをさせている。
再配置が発生する。

Cの関数► _Handle *TSRscGet3*(long Type, int ID);
返り値はリソースへのハンドル。エラーが発生した場合は NULL が返る。

$A4B7 TSResRouteUnLink 2.0

引　数► なし

返り値► D0.L リンクカウント

機　能► コールしたタスクにリンクされているリソース系列をアンリンクする。リンクカウントが 0 になった場合、そのリソース系列は廃棄され、その下のリソースも $A4A4 TSResRemove で削除される。
再配置が発生する。

Cの関数► int *TSResRouteUnLink*(void);
返り値はリンクカウント。

$A4B8 TSMaxIDGet2 2.0

引　数► long Type ；最大 ID を調べるタイプ
long route ；リソース系列

返り値► D0.L Type の最大 ID/リザルトコード

機　能► Type で指定したリソース群のなかで、route で指定したリソース系列のうち最大の ID を返す。

Cの関数► int *TSMaxIDGet2*(long Type, long route);
返り値は、 Type の最大 ID またはリザルトコード。

$A4B9 TSFind 2.0

引　数► long RscHdl ；リソースへのハンドル
long TypePtr ；タイプが格納されるバッファ(1 ロングワード) へのポインタ
long IDPtr ；ID が格納されるバッファ(1 ワード) へのポインタ

返り値► D0.L リソースマップへのハンドル
A0.L リソースマップへのハンドル

機　能► RscHdl で指定したリソースのタイプと ID を調べ、それぞれ TypePtr, IDPtr で示されるバッファに格納する。また、そのリソースが属するリソースマップへのハンドルを返す。
コールしたタスクのリソース系列にリンクされていないリソースを指定した場

合はエラーになる。

Cの関数► Handle *TSFind*(_Handle RscHdl,long *TypePtr,long *IDPtr);
返り値はリソースマップへのハンドル。

$A4BA TSCurResGet2 2.0

引　数► long route ；リソース系列
返り値► D0.L カレントリソースマップへのハンドル
A0.L 最終リソースマップへのハンドル
機　能► route で指定したリソース系列のカレントリソースマップと最終リソースマップを返す。
Cの関数► Handle *TSCurResGet2*(long route);
返り値はカレントリソースマップへのハンドル。

$A4BB TSMaxIDGet3 2.0

引　数► long Type ；リソースタイプ
返り値► D0.L リソース ID
機　能► コールしたタスクの系列、システムリソースの順に検索し、Type で指定したリソース群のなかで最大の ID を返す。
Cの関数► int *TSMaxIDGet3*(long Type);
返り値はリソース ID。

$A4BC TSResFileGet 2.0

引　数► long ResMapHdl ；リソースマップへのハンドル
long buffPtr ；ファイル名（ASCIIZ 型）が格納されるバッファ(90 バイト）へのポインタ
返り値► D0.L ファイルハンドル/リザルトコード
A0.L オープンファイル管理レコードへのポインタ
機　能► ResMapHdl で指定したリソースマップのファイル名を buffPtr に格納する。リソースマップのファイルハンドル、オープンファイル管理レコードへのポインタも返す。
Cの関数► int *TSResFileGet*(Handle ResMapHdl, char *buffPtr);
返り値は、ファイルハンドルまたはリザルトコード。

$A4BD TSResRouteFind 2.0

引　数► long namePtr ；タスク名（ASCIIZ 型）へのポインタ

long taskID ;タスクID
long route ;リソース系列

返り値► D0.L タスクID(下位ワードのみ有効)
< 0 該当するタスクがない

機 能► namePtr で指定した名前、route で指定したリソース系列を持つタスクを、taskID より大きいタスクIDのタスクのなかから探し出す。
namePtr として -1 を指定すると、コールしたタスクの名前で探す。
コールしたタスクは検索の対象とはならない。

Cの関数► long *TSResRouteFind*(const char *namePtr,int taskID,long route);
返り値は、タスクIDまたは結果を意味する数値。

$A4BE TSResTypeList 2.0

引 数► long argc ;タイプの数が返るバッファ(1ロングワード)のアドレス
long argv ;タイプのリスト(タイプの数 × 1ロングワード)へのハンドルが返るバッファのアドレス
long ResMap ;リソースマップのハンドル

返り値► D0.L = 0 正常終了
≠ 0 エラー

機 能► 指定したリソースマップに登録されているタイプの数とリストを得る。
リストはタイプ名(1ロングワード)がタイプの数だけ並んでいる構造で、再配置可能ブロックとして作成される。末尾は 0.L。リストが不要になったら、廃棄する必要がある。
コールしたタスクの、リソース系列にリンクされていないリソースを指定した場合はエラーになる。
再配置が発生する。

Cの関数► int *TSResTypeList*(int *argc,long ***argv,Handle ResMap);
返り値は結果を意味する数値。

$A4BF TSResIDList 2.0

引 数► long argc ;IDの数が返るバッファ(1ロングワード)のアドレス
long argv ;IDのリスト(IDの数 × 1ワード)へのハンドルが返るバッファのアドレス
long ResMapHdl ;リソースマップへのハンドル
long Type ;タイプ

返り値► D0.L　= 0　正常終了
　　　　　　　≠ 0　エラー

機　能► 指定したリソースマップに登録されているタイプの ID の数とリストを得る。
リストは ID(1 ワード) が ID の数だけ並んでいる構造で、再配置可能ブロックとして作成される。末尾は 0.W。リストが不要になったら、廃棄する必要がある。
コールしたタスクのリソース系列にリンクされていないリソースを指定した場合はエラーになる。
再配置が発生する。

Cの関数► int *TSResIDList*(int *argc,short ***argv,Handle ResMapHdl, long Type);
返り値は結果を意味する数値。

$A4C0　TSRscScan2　3.0

引　数►

long	Type	; リソースタイプ
word	startID	; 検索を開始するリソース ID
word	endID	; 検索を終了するリソース ID
word	mode	= 0　:　空いている ID を探す
		≠ 0　:　使用中の ID を探す
long	route	; リソース系列

返り値► D0.L　リソース ID

機　能► Type で指定したリソースタイプについて、route で指定したリソース系列中の startID から endID までの間を調べ、空いている ID (mode が 0 の場合) または使用中の ID (mode が 0 以外の場合) を探す。

Cの関数► int *TSRscScan2*(long Type,int startID,int endID,int mode, long route);
返り値はリソース ID。

ライブラリ　TSSetAbort

引　数►

long	procPtr	; アボート処理ルーチンのアドレス
long	param	; アボート処理ルーチンに渡されるパラメータ

返り値► D0.L　前の処理ルーチンのアドレス

機　能► ハードウェアエラーが発生した場合に実行するアボート処理ルーチンを設定する。
エラーが発生した場合、exit() で終了する前にアボート処理ルーチンが (*procPtr) (-8194, param) の形で呼び出される。
C 言語でコンパイルしたプログラムでのみ有効。

Cの関数► int *TSSetAbort*(void (*procPtr)(), void *param);

プリントマン

#include <PRINT.H>

$A4E0　PMInit

引　数► なし
返り値► D0.L　リザルトコード
機　能► プリントマンを初期化する。
メモリマン、リソースマン、イベントマン、メニューマン、グラフマン、ウィンドウマンが初期化され、リソースファイル SYSTEM.LB がオープンされている必要がある。
Cの関数► int *PMInit*(void);
返り値はリザルトコード。

$A4E1　PMTini

引　数► なし
返り値► D0.L　リザルトコード
機　能► プリントマンの終了処理を行う。
再配置が発生する。
Cの関数► int *PMTini*(void);

$A4E2　PMOpen

引　数► word　drvrID　;ドライバのID
返り値► D0.L　リザルトコード
= -2　すでにドライバがオープンされている
機　能► drvrIDで指定したIDのプリンタドライバを、リソース'PTRD'からメモリ上に読み込み、ロックする。drvrIDとして-1を指定した場合、SRAMに記録されているデフォルトのプリンタドライバが使用される。
再配置が発生する。
Cの関数► int *PMOpen*(int drvrID);
返り値はリザルトコード。

$A4E3　　PMClose

引　数► なし
返り値► D0.L　　リザルトコード
機　能► プリンタドライバの終了処理を行い、ドライバが使用していたブロックを廃棄する。
再配置が発生する。
Cの関数► int *PMClose*(void);
返り値はリザルトコード。

$A4E4　　PMSetDefault

引　数► long　　prRecHdl　　　　;印刷環境レコードへのハンドル
返り値► D0.L　　リザルトコード
A0.L　　印刷環境レコードへのハンドル
機　能► prRecHdl で指定した印刷環境レコードにデフォルトの値（リソース 'PrEV' の ID 0 に記録されている、あるいはドライバ自体が保持している）をセットする。
再配置が発生する。
Cの関数► int *PMSetDefault*(Print **prRecHdl);
返り値はリザルトコード。

$A4E5　　PMValidate

引　数► long　　prRecHdl　　　　;印刷環境レコードへのハンドル
返り値► D0.L

0	調整せず、レコードの内容に変化はない
1	調整を行った
-1	エラー

機　能► prRecHdl で指定した印刷環境レコードの内容が正しいかどうかをチェックし、調整する。
Cの関数► int *PMValidate*(Print **prRecHdl);
返り値は調整結果を意味する数値。

$A4E6　　PMImageDialog

引　数► long　　prRecHdl　　　　;印刷環境レコードへのハンドル
返り値► D0.L

0	レコードの内容に変化はない
1	設定を行った
-1	エラー

機　能► ページ印刷用の印刷環境設定ダイアログをオープンし、ユーザの操作を受け付けたあと、クローズする。その結果をもとにprRecHdlで指定した印刷環境レコードの内容を設定する。
再配置が発生する。

Cの関数► int *PMImageDialog*(Print **prRecHdl);
返り値は設定結果を意味する数値。

$A4E7　PMStrDialog

引　数► long　prRecHdl　；印刷環境レコードへのハンドル

返り値► D0.L

0	レコードの内容に変化はない
1	設定を行った
-1	エラー

機　能► コード印刷用の印刷環境設定ダイアログをオープンし、ユーザの操作を受け付けたあと、クローズする。その結果をもとにprRecHdlで指定した印刷環境レコードの内容を設定する。
再配置が発生する。

Cの関数► int *PMStrDialog*(Print **prRecHdl);
返り値は設定結果を意味する数値。

$A4E8　PMJobDialog　3.0

引　数► long　prRecHdl　；印刷環境レコードへのハンドル

返り値► D0.L

1	設定
2	取消
-1	エラー

機　能► 印刷開始・終了ページ、印刷枚数の設定、カラー印刷、ドラフト印刷に関する設定を行うダイアログを表示する。

Cの関数► int *PMJobDialog*(Print **prRecHdl);
返り値は結果を意味する数値。

$A4E9　PMEnvCopy

引　数► long　srcHdl　；コピー元の印刷環境レコードへのハンドル
long　dstHdl　；コピー先の印刷環境レコードへのハンドル

返り値► D0.L

0	調整を行わなかった
1	調整を行った
-1	エラー

機　能► srcHdlで指定した印刷環境レコードの内容をdstHdlで指定したレコードにコピーする。その際、値のチェックと調整が行われる。

Cの関数► int *PMEnvCopy*(Print **srcHdl, Print **dstHdl);
返り値は調整結果を意味する数値。

$A4EA　PMJobCopy

引　数► long　srcHdl　；コピー元の印刷環境レコードへのハンドル
long　dstHdl　；コピー先の印刷環境レコードへのハンドル

返り値► D0.L

0	調整を行わなかった
1	調整を行った
-1	エラー

機　能► srcHdl で指定した印刷環境レコードの実行部分のデータを、dstHdl で指定したレコードにコピーする。その際、値のチェックと調整が行われる。
実行部分とは、具体的には印刷開始ページ（prFstPage）、印刷終了ページ（prLstPage）、1 ページあたりの印刷枚数（prDupPage）、印刷モード（prMode）、印刷モードのマスク（prMask）、そしてシステム予約（prJobRsv）を意味する。

Cの関数► int *PMJobCopy*(Print **srcHdl, Print **dstHdl);
返り値は調整結果を意味する数値。

$A4EB　PMOpenImage

引　数► long　prRecHdl　；印刷環境レコードへのハンドル

返り値► D0.L　リザルトコード
A0.L　グラフレコードへのポインタ

機　能► ページ印刷用のグラフレコードを作成し、ページ印刷を開始する。
再配置が発生する。

Cの関数► Graph **PMOpenImage*(Print **prRecHdl);
返り値はグラフレコードへのポインタ。

$A4EC　PMRecordPage

引　数► long　rectPtr　；印刷範囲を意味するレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード

機　能► ページ印刷のスクリプトの記録を開始する。rectPtr で指定した範囲があとで印刷される。
再配置が発生する。

Cの関数► int *PMRecordPage*(Rect *rectPtr);
返り値はリザルトコード。

$A4ED　PMPrintPage

引　数► long　param　; 必ず0を指定する
返り値► D0.L　リザルトコード
機　能► ページ印刷用スクリプトの記録を終了し、実際の印刷を開始する。
再配置が発生する。
Cの関数► int *PMPrintPage*(void);
引数を指定する必要はない。
返り値はリザルトコード。

$A4EE　PMCancelPage

引　数► なし
返り値► D0.L　リザルトコード
機　能► ページ印刷用スクリプトの記録を中止する。印刷は行われない。
再配置が発生する。
Cの関数► int *PMCancelPage*(void);
返り値はリザルトコード。

$A4EF　PMAction

引　数► word　ctrl　; 動作の指定

0(PC_STAT)	印刷を続行する
1(PC_END)	印刷を終了する
2(PC_STOP)	印刷を中断する
3(PC_CONT)	印刷を再開する

返り値► D0.L　結果

0(P_FINISH)	印刷が終了した
1(P_WORKING)	印刷中
2(P_RESTING)	印刷を中断した
3(P_TIMEOUT)	タイムアウト発生
-1(P_ERROR)	エラー発生

機　能► 印刷処理を行う。ctrlで指定した動作を行い、その結果を返す。
再配置が発生する。
Cの関数► int *PMAction*(int ctrl);
返り値は実行結果。

$A4F0　PMCloseImage

引　数► なし

返り値► D0.L　　リザルトコード
機　能► ページ印刷を終了し、グラフレコードなどを廃棄する。
再配置が発生する。
Cの関数► int *PMCloseImage*(void);
返り値はリザルトコード。

$A4F1　PMDrawString

引　数► long　prRecHdl　　; 印刷環境レコードへのハンドル
long　strHdl　　; 文字列へのハンドル
long　length　　; 文字列のバイト数
long　strOpt　　; 印刷オプション

0	印刷終了時に改ページコードを出力する
1	印刷終了時に改ページコードを出力しない

返り値► D0.L　　リザルトコード
機　能► strHdl, length で指定した文字列のコード印刷を開始する。
再配置が発生する。
Cの関数► int *PMDrawString*(Print **prRecHdl, char **strHdl,
long length, int strOpt);
返り値はリザルトコード。

$A4F2　PMVer

引　数► なし
返り値► D0.L　　バージョン番号（バージョン 1.00 で$0100）
機　能► プリントマンのバージョンを返す。
Cの関数► int *PMVer*(void);
返り値はバージョン番号。

$A4F3　PMDrvrVer

引　数► なし
返り値► D0.L　　バージョン番号（バージョン 1.00 で$0100）
= -1　プリンタドライバがオープンされていない
機　能► 現在オープンされているプリンタドライバのバージョンを返す。
Cの関数► int *PMDrvrVer*(void);
返り値はバージョン番号。

$A4F4　PMDrvrCtrl

引　数► long　cmd　;ドライバに与えるコマンド
long　param1　;パラメータ1
long　param2　;パラメータ2
long　param3　;パラメータ3

返り値► D0.L　リザルトコード

機　能► プリンタドライバを直接制御する。プリンタドライバに与えるコマンド、パラメータについては『追補版 SX-WINDOW プログラミング』本文参照。

Cの関数► int *PMDrvrCtrl*(int cmd, ...);
返り値はリザルトコード。

$A4F5　PMDrvrID

引　数► なし

返り値► D0.L　プリンタドライバの ID(下位ワードのみ意味を持つ)
= -1　プリンタドライバがオープンされていない

機　能► 現在オープンされているプリンタドライバの ID を返す。

Cの関数► int *PMDrvrID*(void);
返り値はプリンタドライバの ID。

$A4F6　PMDrvrHdl

引　数► なし

返り値► D0.L　リザルトコード
= -1　プリンタドライバがオープンされていない
A0.L　プリンタドライバへのハンドル

機　能► 現在オープン中のプリンタドライバが納められているブロックへのハンドルを返す。

Cの関数► Handle *PMDrvrHdl*(void);
返り値はプリンタドライバへのハンドル。
エラーの場合は NULL が返る。

$A4F7　PMMaxRect

引　数► long　prRecHdl　;印刷環境レコードへのハンドル
word　pKind　;用紙の種類
long　rectPtr　;結果が返るレクタングルレコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　印刷環境レコードへのハンドル
機　能► pKind で指定した用紙の印刷可能な最大範囲を、rectPtr で指定したレクタングルレコードに返す。
Cの関数► int *PMMaxRect*(Print **prRecHdl, int pKind, Rect *rectPtr);
返り値はリザルトコード。

$A4F8　PMSaveEnv

引　数► long　prRecHdl　；印刷環境レコードへのハンドル
返り値► D0.L　リザルトコード
機　能► prRecHdl で指定した印刷環境レコードの内容をデフォルトの値としてリソース‘PrEV’の ID 0 に記録する。
再配置が発生する。
Cの関数► int *PMSaveEnv*(Print **prRecHdl);
返り値はリザルトコード。

$A4F9　PMReady

引　数► なし
返り値► D0.L　プリンタの状態

0(PS_BUSY)	印刷不可
1(PS_READY)	印刷可

機　能► プリンタの状態を調べ、結果を返す。
Cの関数► int *PMReady*(void);
返り値はプリンタの状態を意味する数値。

$A4FA　PMProcPrint

引　数► long　prRecHdl　；印刷環境レコードへのハンドル
long　procPtr　；ユーザプロセスのアドレス
返り値► D0.L　リザルトコード
機　能► procPtr で指定したユーザプロセスを登録し、プロセス印刷を開始する。
Cの関数► int *PMProcPrint*(Print **prRecHdl,
int (*procPtr)(Print **prRecHdl, Rect *rectPtr));
返り値はリザルトコード。

$A4FB　PMDrvrInfo

引　数► word　drvrID　;プリンタドライバのID
long　prdrInfoPtr　;プリンタドライバ情報レコードへのポインタ

返り値► D0.L　リザルトコード
A0.L　prdrINfoPtr
(A0).w　プリンタドライバのID
2(A0).w　プリンタドライバのバージョン
4(A0)　プリンタ名(ASCIIZ)

機　能► drvrIDで指定したプリンタドライバに関する情報を、prdrInfoPtrで指定したプリンタドライバ情報レコードに返す。drvrIDとして-1を指定すると、デフォルトのプリンタドライバの情報が返る。

Cの関数► int *PMDrvrInfo*(int drvrID, PrtDInfo *prdrInfoPtr);
返り値はリザルトコード。

$A4FC　PMGetDefDlog　3.0

引　数► word　dKind　;ダイアログの種類

0(PDL_IMG)	イメージダイアログ
1(PDL_JOB)	ジョブダイアログ

返り値► D0.L

1	正常終了
-1	エラー

A0.L　デフォルトルーチンのアドレス

機　能► $A4E6 PMImageDialog、または$A4E8 PMJobDialogで表示されるダイアログの処理を行うデフォルトのルーチンのアドレスを返す。

Cの関数► PrDlog **PMGetDefDlog*(int dKind);
返り値はデフォルトルーチンのアドレス。
エラーが発生した場合、NULLが返る。

$A4FD　PMSetRange　3.0

引　数► long　prRecHdl　;印刷環境レコードへのハンドル
word　topRange　;印刷開始ページ数
word　endRange　;印刷終了ページ数

返り値► D0.L

1	正常終了
-1	エラー

機　能► prRecHdlで指定した印刷環境レコードに、topRangeとendRangeで指定した印刷開始ページと印刷終了ページを設定する。
topRangeはprMinPageに、endRangeはprMaxPageに格納される。

Cの関数► int *PMSetRange*(PrRec **prRecHdl, int topRange, int endRange);
返り値は結果を意味する数値。

$A4FE PMPutID 3.0

引　数► word CPDFID ；カラー変換モジュールの ID

返り値► D0.L 前のカラー変換モジュールの ID
= -1 エラー

機　能► 印刷時に使用するデフォルトのカラー変換モジュールの ID として CPDFID をセットする。この値は、リソース‘PrEV’の ID 1 に記録される。
CPDFID として-1 を指定すると、現在設定されている ID を返すのみとなる。

Cの関数► int *PMPutID*(int CPDFID);
返り値はカラー変換モジュールの ID。

フォントマン

#include <SXGRAPH.H>

$A500　FMInit　2.0

引　数► なし

返り値► D0.L

0	常駐している
-1	常駐していない

機　能► IFM.X の常駐チェックを行う。フォントマンの初期化は行わない。

Cの関数► int *FMInit*(void);

返り値は常駐チェックの結果。

$A501　FMTini　2.0

引　数► なし

返り値► D0.L

0	常駐している
-1	常駐していない

機　能► 使用禁止コール。

IFM.X の終了処理を行うが、常駐解除されるわけではない。

$A502　FMGetFontList　2.0

引　数► なし

返り値► D0.L　フォントリストへのポインタ

機　能► フォントマンに登録されているフォントのリストを返す。フォントリストの構造は以下のとおり。

```
typedef struct FontList {
short    id;             /* フォント ID */
short    type;           /* フォントタイプ 全角:1 半角:0 */
short    motherID;       /* マザー ID */
char     FontName[26];   /* フォント名 */
} FontList;
```

フォントリストが必要なくなった場合、$A02F MMPtrDispose で廃棄する必要がある。再配置が発生する。

Cの関数► FontList **FMGetFontList*(void);

返り値はフォントリストへのポインタ。

$A503　FMSetChacheSize　2.0

引　数► long　chacheSize　;フォントキャッシュのサイズ(Kバイト単位)
返り値► D0.L　変更後のフォントキャッシュのサイズ(Kバイト単位)
機　能► フォントキャッシュのサイズを cacheSize で指定した値に設定する。フォントキャッシュの内容は失われる。
再配置が発生する。
Cの関数► int *FMSetCacheSize*(int cacheSize);
返り値はフォントキャッシュのサイズ。

$A504　FMGetChacheSize　2.0

引　数► なし
返り値► D0.L　フォントキャッシュのサイズ(Kバイト単位)
機　能► 現在のフォントキャッシュのサイズを返す。
Cの関数► int *FMGetCacheSize*(void);
返り値はフォントキャッシュのサイズ。

$A505　FMSetSpaceWidth　2.0

引　数► long　spaceWidth　;スペースの幅(固定小数点数)
返り値► D0.L　現在のスペースの幅(固定小数点数)
機　能► スペースの幅を、spaceWidth で指定した値に設定する。spaceWidth は文字の縦サイズとの比率を固定小数点数で表現したもので、整数部22ビット、小数部10ビット(つまり、倍率を1024倍した数を指定する)。デフォルトは256。負の値も指定可能。
半角スペースは spaceWidth で指定した幅に、全角スペースはその倍の幅になる。
Cの関数► int *FMSetSpaceWidth*(int spaceWidth);
返り値はスペースの幅。

$A506　FMGetSpaceWidth　2.0

引　数► なし
返り値► D0.L　現在のスペースの幅(固定小数点数)
機　能► 現在のスペースの幅を返す。

Cの関数► int *FMGetSpaceWidth*(void);
現在のスペースの幅。

$A507　FMSetTracking　2.0

引　数► long　trackWidth　;トラッキング幅
返り値► D0.L　現在のトラッキング幅
機　能► トラッキング幅をtrackWidthで指定した値に設定する。trackWidthはドット単位で指定し、負の数も指定可能。
Cの関数► int *FMSetTracking*(int trackWidth);
返り値は現在のトラッキング幅。

$A508　FMGetTracking　2.0

引　数► なし
返り値► D0.L　現在のトラッキング幅
機　能► 現在のトラッキング幅を返す。
Cの関数► int *FMGetTracking*(void);
返り値は現在のトラッキング幅。

$A509　FMGetKerningWidth　2.0

引　数► word　beforeChar　;直前の文字のコード
word　curChar　;現在の文字のコード
返り値► D0.L　2文字間のカーニング幅
機　能► beforeCharとcurCharの2つの英文字間のカーニング幅を返す。カレントグラフレコードのフォントカインド、フォントサイズが参照され、カーニング情報が用意されているフォントを使用している場合にのみ正しい値が返る。
Cの関数► int *FMGetKerningWidth*(unsigned short beforeChar, unsigned short curChar);
返り値はカーニング幅。

$A50A　FMFontMenuSelect　2.0

引　数► long　pt　;右ボタンが押された座標（グローバル座標系）
返り値► D0.L　選択されたフォントのID
機　能► ptで指定したポイントから現在登録されているフォントを選択するメニューを表示し、選択されたフォントのID(＝フォントカインド）を返す。

再配置が発生する。

Cの関数► int *FMFontMenuSelect*(LPoint pt);
返り値は選択されたフォントのID。

$A50B　FMGetFontPolyData　2.0

引　数► word　ch　; 文字のコード

返り値► D0.L　ニューポリゴンレコードへのポインタ

機　能► ch で指定した文字をニューポリゴン形式に変換して返す。カレントグラフレコードのフォントカインド、フォントサイズが参照される。
再配置が発生する。

Cの関数► NPoly **FMGetFontPolyData*(int ch);
返り値はニューポリゴンレコードへのポインタ。

セマフォマン

$A5F0　SXInitSemaphore　3.0

引　数► なし

返り値► なし

機　能► セマフォ管理テーブルを初期化する。
SX シェルが初期化するので、シェル上で動作するアプリケーションは使用することができない。

Cの関数► void *SXInitSemaphore*(void);
返り値はない。

$A5F1　SXAddSemaphore　3.0

引　数► long　key　; セマフォのキーとなる ASCIIZ 文字列へのポインタ

返り値► D0.L

0	すでにそのキーは登録済み
< 0	エラー
other	セマフォの ID

機　能► ASCIIZ 文字列 key をキーとして登録する。

Cの関数► int *SXAddSemaphore*(char *key);
返り値は結果を意味する数値。

$A5F2　SXDelSemaphore　3.0

引　数► long　key　; キーとなる ASCIIZ 文字列へのポインタ
long　ID　; $A5F1 SXAddSemaphore で得られたセマフォの ID

返り値► D0.L

1	削除成功
0	キーは未登録
< 0	エラー

機　能► ASCIIZ 文字列 key で指定したキーを削除する。

Cの関数► int *SXDelSemaphore*(char *key, int ID);
返り値は結果を意味する数値。

$A5F3　SXFindSemaphore　3.0

引　数► long　key　; キーとなる ASCIIZ 文字列へのポインタ

返り値► D0.L

1	登録されている
0	登録されていない

機　能► key で指定した文字列が登録されているかどうかを調べる。

Cの関数► int *SXFindSemaphore*(char *key);

返り値は結果を意味する数値。

カラーマン

$A600　CLInit　3.0

引　数► なし

返り値► D0.L

1	正常終了
-1	エラー

機　能► カラーマンの初期化を行う。
メモリマン、リソースマン、グラフマンが初期化されている必要がある。

Cの関数► int *CLInit*(void);
返り値は結果を意味する数値。

$A601　CLTini　3.0

引　数► なし

返り値► D0.L

1	正常終了
-1	エラー

機　能► カラーマンの終了処理を行う。

Cの関数► int *CLTini*(void);
返り値は結果を意味する数値。

$A602　CLNewPalet　3.0

引　数► word　pKind　; パレットの種類

0 (CLK_WITHLIST)	カラーリストあり
1 (CLK_NOLIST)	カラーリストなし

long　maxNum　; カラーの数 (1～1024)

返り値► D0.L

1	正常終了
-1	エラー

A0.L　パレットハンドル
エラーの場合は 0

機　能► pKind で指定した種類で、maxNum で指定した数の色を表現するための新しいパレットハンドルを作成する。
作成されたパレットハンドルには、以下のような値が設定される。

paletDev　0(CDEV_NONE) デバイスにリンクされていない

paletDepth -1	ピクセルの深さは X680x0 形式 (5+5+5+1 bit)
paletOrder -1	ピクセルの並びは X680x0 形式 (G+R+B+i)

「カラーリストあり」を指定した場合、すべてのカラーのカラーレコードには以下のような値が設定される。

ciRGB	すべての要素が $0000
ciLevel	$0000
ciRange	$2000

再配置が発生する。

Cの関数► Palet ***CLNewPalet*(int pKind, int maxNum);
返り値はパレットハンドル。
エラーが発生した場合、NULL が返る。

$A603 CLRefer 3.0

引　数► word rscID ; パレットリソース ('PALT') のリソース ID

返り値► D0.L

1	正常終了
-1	エラー

A0.L パレットハンドル
エラーの場合は 0

機　能► rscID で指定したパレットリソースを参照し、パレットハンドルを作成する。
SYSTEM.LB に用意されているパレットリソースは以下のとおり。

-128	(PALT_G16)	グラフィック 16 色
-129	(PALT_G256)	グラフィック 256 色
-130	(PALT_G65536)	グラフィック 65536 色
1	(PALT_TEXT)	テキストタイプ
3	(PALT_MONO)	モノクロ
4	(PALT_RGB8)	RGB8 色
16		32 色 (未使用)
17		64 色 (未使用)
18		128 色 (未使用)

再配置が発生する。

Cの関数► Palet ***CLRefer*(int rscID);
返り値はパレットハンドル。
エラーが発生した場合、NULL が返る。

$A604 CLDupDevicePalet 3.0

引　数► word cDevID ; カラーデバイス ID

-1	(CDEV_USER)	ユーザ定義
0	(CDEV_NONE)	デバイスにリンクされていない
1	(CDEV_TEXT)	テキスト画面
2	(CDEV_GRAPH)	グラフィック画面
3	(CDEV_MONO)	モノクロ
4	(CDEV_RGB8)	RGB8 色

返り値► D0.L

1	正常終了
-1	エラー

A0.L　パレットハンドル
エラーの場合は0

機　能► cDevID で指定したデバイスパレットと同じ内容のパレットハンドルを作成する。
再配置が発生する。

Cの関数► Palet ***CLDupDevicePalet*(int cDevID);
返り値はパレットハンドル。
エラーが発生した場合 NULL が返る。

$A605　CLDisposePalet　3.0

引　数► long　paletHdl　; パレットハンドル

返り値► D0.L

1	正常終了
-1	エラー

機　能► paletHdl で指定したパレットハンドルを廃棄する。
このコールのあと、割り付け変更イベントを登録する。

Cの関数► int *CLDisposePalet*(Palet **paletHdl);
返り値は結果を意味する数値。

$A606　CLSetCInfo　3.0

引　数► long　paletHdl　; パレットハンドル
long　flags　; 設定を行う要素を意味するフラグの集合体

bit0	赤
bit1	緑
bit2	青
bit3	アルファ
bit4	優先レベル
bit5	レンジ

long　palNo　; 設定状況を行うパレット番号
long　number　; 設定状況を行うパレット数

long　cInfoPtr　；設定状況する情報を収めたカラー情報レコードへのポインタ

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdl で指定したパレットハンドル中の、palNo から始まる number 個のパレットに対して、cInfoPtr で指定した情報の flags で示された要素を設定する。
パレットの割り付けに変更が生じる場合がある。

Cの関数► int *CLSetCInfo*(Palet **paletHdl, int flags, int palNo, int number, CInfo *cInfoPtr);
返り値は結果を意味する数値。

$A607　CLGetCInfo　3.0

引　数► long　paletHdl　；パレットハンドル
long　flags　；設定状況を得る要素を意味するフラグの集合体

bit0	赤
bit1	緑
bit2	青
bit3	アルファ
bit4	優先レベル
bit5	レンジ

long　palNo　；設定状況を得るパレット番号
long　number　；設定状況を得るパレット数
long　cInfoPtr　；設定する情報を格納するカラー情報レコードへのポインタ

返り値► D0.L

1	正常終了
-1	エラー

機　能► paletHdl で指定したパレットハンドル中の、palNo から始まる number 個のパレットについて、flags で示された要素の設定状況を cInfoPtr で指定したカラー情報レコードに格納する。
number として 2 以上を指定した場合、最後のパレットに関する情報がカラー情報レコードに格納される。

Cの関数► int *CLGetCInfo*(Palet **paletHdl, int flags, int palNo, int number, CInfo *cInfoPtr);
返り値は結果を意味する数値。

$A608 CLAlloc 3.0

引　数► long　paletHdl　;パレットハンドル

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdlで指定したパレットハンドル中の、デバイスに割り付けられていないカラーの割り付けを行う。ガイドパレット以外を指定した場合、エラーが発生する。
パレットの割り付けに変更が生じる可能性がある。

Cの関数► int *CLAlloc*(Palet **paletHdl);
返り値は結果を意味する数値。

$A609 CLAllocOne 3.0

引　数► long　paletHdl　;パレットハンドル
long　palNo　;パレット番号

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdlで指定したパレットハンドル中の、palNoで指定したパレットについてカラーの割り付けを行う。すでに割り付けられていた場合は何もしない。ガイドパレット以外を指定した場合、エラーが発生する。
パレットの割り付けに変更が生じる可能性がある。

Cの関数► int *CLAllocOne*(Palet **paletHdl, int palNo);
返り値は結果を意味する数値。

$A60A CLFree 3.0

引　数► long　paletHdl　;パレットハンドル

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdlで指定したパレットハンドル中の、デバイスに割り付けられているカラーの割り付けを解除する。ガイドパレット以外を指定した場合、エラーが発生する。
このコールのあと、割り付け変更イベントを登録する。
パレットの割り付けに変更が生じる可能性がある。

Cの関数► int *CLFree*(Palet **paletHdl);
返り値は結果を意味する数値。

$A60B　CLFreeOne　3.0

引　数► long　paletHdl　;パレットハンドル
long　palNo　;パレット番号

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdlで指定したパレットハンドル中の、palNoで指定したパレットについてカラーの割り付けを解除する。割り付けられていなかった場合は何もしない。ガイドパレット以外を指定した場合、エラーが発生する。
このコールのあと、割り付け変更イベントを登録する。
パレットの割り付けに変更が生じる可能性がある。

Cの関数► int *CLFreeOne*(Palet **paletHdl, int palNo);
返り値は結果を意味する数値。

$A60C　CLActive　3.0

引　数► long　paletHdl　;パレットハンドル

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdlで指定したパレットハンドルのカラーをデバイスパレットに再割り付けする。アクティベートイベントの処理中で、自分のウィンドウがアクティブになった場合に使用する。
自分のパレットに割り付けの変更があった場合は、再描画を行う必要がある。
ガイドパレット以外を指定した場合はエラーが発生する。
このコールのあと、割り付け変更イベントを登録する。
パレットの割り付けに変更が生じる可能性がある。

Cの関数► int *CLActive*(Palet **paletHdl);
返り値は結果を意味する数値。

$A60D　CLRealloc　3.0

引　数► long　paletHdl　;パレットハンドル

返り値► D0.L

0	パレットの割り付けに変更がなかった
1	自分に対し、パレットの割り付けの変更があった
2	自分以外に対し、パレットの割り付けの変更があった
-1	エラー

機　能► paletHdl で指定したパレットハンドルのカラーをデバイスパレットに再割り付けする。システムイベントの処理中で、割り付け変更イベントを受け取った場合に使用する。

パレットの割り付けに変更が生じる可能性がある。

Cの関数► int *CLRealloc*(Palet **CLRealloc);

返り値は結果を意味する数値。

$A60E　CLLinkPalet　3.0

引　数► long　paletHdl　; パレットハンドル

word　cDevID　; カラーデバイス ID

-1(CDEV_USER)	ユーザ定義
0(CDEV_NONE)	デバイスにリンクされていない
1(CDEV_TEXT)	テキスト画面
2(CDEV_GRAPH)	グラフィック画面
3(CDEV_MONO)	モノクロ
4(CDEV_RGB8)	RGB8 色

返り値► D0.L

1	正常終了
-1	エラー

機　能► paletHdl で指定したパレットを cDevID で指定したデバイスにリンクし、ガイドパレットにする。

再配置が発生する。

Cの関数► int *CLLinkPalet*(Palet **paletHdl, int cDevID);

返り値は結果を意味する数値。

$A60F　CLUnlinkPalet　3.0

引　数► long　paletHdl　; パレットハンドル

返り値► D0.L

1	正常終了
-1	エラー

機　能► paletHdl で指定したパレットハンドルのリンク状態を解除する

Cの関数► int *CLUnlinkPalet*(Palet **paletHdl);

返り値は結果を意味する数値。

$A610　CLSetDeviceMode　3.0

引　数► word　cDevID　; デバイス ID
word　mode　; モード
bit0　変更不可

返り値► D0.L　前のモード
= -1　エラー

機　能► cDevID で指定したデバイスのモードを mode にする。

Cの関数► int *CLSetDeviceMode*(int cDevID, int mode);
返り値は前のモード。

$A611　CLGetDeviceMode　3.0

引　数► word　cDevID　; デバイス ID

返り値► D0.L　モード
= -1　エラー

機　能► cDevID で指定したデバイスのモードを返す。

Cの関数► int *CLGetDeviceMode*(int cDevID);
返り値はデバイスのモード。

$A612　CLGetDevice　3.0

引　数► word　cDevID　; デバイス ID

返り値► D0.L

1	正常終了
-1	エラー

A0.L　デバイスパレットハンドル

機　能► cDevID で指定したデバイスのデバイスパレットハンドルを返す。

Cの関数► Palet ***CLGetDevice*(int cDevID);
返り値はデバイスパレットハンドル。
エラーが発生した場合、 NULL が返る。

$A613　CLAddDevice　3.0

引　数► long　paletHdl　; パレットハンドル
word　cDevID　; デバイス ID

返り値► D0.L　登録したデバイス ID
= -1　エラー

機　能► paletHdl で指定したパレットハンドルを、cDevID で指定したデバイス ID を持つデバイスパレットとしてシステムに登録する。

cDevID として-1 を指定すると、新しいデバイス ID が割り当てられる。
C 言語からコールする場合は、ライブラリの CLAddDevice を利用すること。

$A614 CLDelDevice 3.0

引　数► word cDevID ;デバイス ID

返り値► D0.L

1	正常終了
-1	エラー

機　能► cDevID で指定したデバイスパレットを削除する。
cDevID として指定できるのは、自分で登録したデバイスパレットのデバイス ID に限られる。

Cの関数► int *CLDelDevice*(int cDevID);
返り値は結果を意味する数値。

$A615 CLSetDeviceRGB 3.0

引　数► word cDevID ;デバイス ID
long palNo ;設定を行うパレット番号
long number ;設定を行うパレット数
long cListPtr ;カラーレコードの配列へのポインタ

返り値► D0.L

1	正常終了
-1	エラー

機　能► cDevID で指定したデバイスパレット中の、palNo から始まる number 個のパレットについて、cListPtr で指定したカラーレコードの配列の内容を設定する。
デバイス ID が 1 (CDEV_TEXT) または 2 (CDEV_GRAPH) の場合は、ハードウェアパレットも設定される。

Cの関数► int *CLSetDeviceRGB*(int cDevID, int palNo, int number, cRGB *cListPtr);
返り値は結果を意味する数値。

$A616 CLDupPalet 3.0

引　数► long paletHdl ;パレットハンドル

返り値► D0.L

1	正常終了
-1	エラー

A0.L 複製されたパレットハンドル

機　能► paletHdl で指定したパレットハンドルを複製する。
疑似ハンドルも可。

再配置が発生する。

Cの関数► Palet **$CLDupPalet$(Palet **paletHdl);

返り値はパレットハンドル。

エラーが発生した場合、NULL が返る。

$A617 CLCopyPalet 3.0

引　数► long dPaletHdl ; コピー先のパレットハンドル

long sPaletHdl ; コピー元のパレットハンドル

返り値► D0.L

1	正常終了
−1	エラー

A0.L コピーされたパレットハンドル

機　能► sPaletHdl で指定したパレットハンドルのカラー情報を、dPaletHdl で指定したパレットハンドルにコピーする。

どちらか一方のパレットハンドルにカラーリストが存在しない場合はエラーとなる。

コピー元よりコピー先のほうが色数が多い場合は、コピー元の色の数だけコピーされ、あとは変化しない。

コピー元よりコピー先のほうが色数が少ない場合は、コピー先の色の数だけコピーされる。

Cの関数► int $CLCopyPalet$(Palet **dPaletHdl, Palet **sPaletHdl);

返り値は結果を意味する数値。

$A618 CLSetPickEntry 3.0

引　数► long cPickProc ; カラーピックアップルーチンへのポインタ

返り値► D0.L 前のカラーピックアップルーチンへのポインタ

= -1 エラー

A0.L 前のカラーピックアップルーチンへのポインタ

機　能► カラーピックアップルーチンを登録する。

カラーピックアップルーチンの仕様は以下のとおり。

int CPickUp(cPick *pBlock)

引数

```
typedef struct {
        cInfo       *sInfo; /* 元のカラーレコードへのポインタ */
        cInfo       *cList; /* サーチする cInfo 配列へのポインタ */
        int         listNum;/* サーチする cInfo 配列の要素数 */
        /* 結果を格納するカラーレコードへのポインタ */
        cInfo       *dInfo;
} cPick;
cPick *pBlock;
```

返り値

= 1 (CLS_COMPLETE)　sInfo と同じ色を発見した

= 2 (CLS_INCOMPLETE) sInfo に近い色で代用した

カラーピックアップルーチンは、sInfo に最も近い色を cList のなかから探して dInfo に格納する。カラーマンが色の割り付けを行う際に呼び出される。

Cの関数► PickUp *CLSetPickEntry*(PickUp (*cPickProc)());

返り値は前のカラーピックアップルーチンへのポインタ。

$A619　CLValueToRGB　3.0

引　数► word　order　; ピクセルの RGB の並び方

0 (CLO_RGB)	R、G、B の順
1 (CLO_RBG)	R、B、G の順
2 (CLO_GRB)	G、R、B の順
3 (CLO_GBR)	G、B、R の順
4 (CLO_BRG)	B、R、G の順
5 (CLO_BGR)	B、G、R の順

word　depth　; ピクセルの深さ

-1 : X680x0 形式

long　rgbRecPtr　; 結果が返る RGB レコードへのポインタ

long　value　; カラーコード

返り値► D0.L　=1　つねに 1

機　能► order と depth で示された形式のカラーコード value を RGB 形式に変換し、rgbRecPtr で指定された RGB レコードに格納する。

depth として-1 を指定した場合、X680x0 形式のカラーコードを意味する。このとき、order は意味を持たない。

Cの関数► int *CLValueToRGB*(int order, int depth, cRGB *rgbRecPtr, int value);

結果は cRGB 型の変数 rgbRecPtr に返る。

返り値はつねに 1。

$A61A　CLRGBToValue　3.0

引　数► word　order　; ピクセルの RGB の並び方

0 (CLO_RGB)	R、G、B の順
1 (CLO_RBG)	R、B、G の順
2 (CLO_GRB)	G、R、B の順
3 (CLO_GBR)	G、B、R の順
4 (CLO_BRG)	B、R、G の順
5 (CLO_BGR)	B、G、R の順

word　depth　; ピクセルの深さ

-1 : X680x0 形式

long rgbRecPtr ;RGB レコードへのポインタ

返り値► D0.L カラーコード

機 能► rgbRecPtr で指定した RGB レコードの内容を、order と depth で示される形式のカラーコードに変換する。

depth として-1 を指定した場合、X680x0 形式のカラーコードを返す。このとき order は意味を持たない。

Cの関数► int *CLRGBToValue*(int order,int depth,cRGB *rgbRecPtr);
返り値はカラーコード。

$A61B CLRefer2 3.0

引 数► word pltlID ;パレットテンプレートリソース（'PLTL'）の ID

返り値► D0.L

1	正常終了
-1	エラー

A0.L パレットハンドル

機 能► pltlID で指定したパレットテンプレートを参照し、パレットハンドルを作成する。

パレットテンプレートはパレットリソース（またはデバイスパレット）と、そのなかの必要な色を指定するリソースで、SYSTEM.LB に用意されているパレットテンプレートリソースは以下のとおり。

-128	(PALT_G16)	グラフィック 16 色
-129	(PALT_G256)	グラフィック 256 色
-130	(PALT_G65536)	グラフィック 65536 色
-11	(PALT_PAT1)	PAT1 表示用
-12	(PALT_PAT2)	PAT2 表示用
-13	(PALT_PAT3)	PAT3 表示用
-14	(PALT_PAT4)	PAT4 表示用
-22		未使用
-23		未使用
-24		未使用
-33		未使用
-34		未使用
-44		未使用
3	(PALT_MONO)	モノクロ
4	(PALT_RGB8)	RGB8 色

以下の 2 つの ID は、リソースとしては用意されておらず、指定した場合はシステム内部で特殊な処理が行われる。

1	(PLTL_TEXT)	テキスト自動判別
2	(PLTL_GRAPH)	グラフィック自動判別

再配置が発生する。

Cの関数► Palet ***CLRefer2*(int pltlID);

返り値はパレットハンドル。
エラーが発生した場合、NULL が返る。

$A61C CLLoadText 3.0

引 数► なし

返り値► D0.L

1	正常終了
-1	エラー

機 能► テキストのデバイスパレットを初期化する。

Cの関数► int *CLLoadText*(void);
返り値は結果を意味する数値。

$A61D CLValueToPalet 3.0

引 数► long vListPtr ;カラーコードの配列へのポインタ
long vNumber ;カラーコードの配列の要素数

返り値► D0.L

1	正常終了
-1	エラー

A0.L パレットハンドル

機 能► vNumber 個の要素を持つ vListPtr で指定したカラーコードの配列から、パレットハンドル(カラーリスト付き)を作成する。
カラーコードの配列は、カラーコード(word)を並べたもの。
再配置が発生する。

Cの関数► Palet ***CLValueToPalet*(VList *vListPtr, int vNumber);
返り値はパレットハンドル。
エラーが発生した場合、 NULL が返る。

$A61E CLPaletToValue 3.0

引 数► long paletHdl ;パレットハンドル
long vListPtr ;結果が返るカラーコードの配列へのポインタ

返り値► D0.L

1	正常終了
-1	エラー

A0.L 結果が格納されたカラーコードの配列へのポインタ

機 能► paletHdl で指定したパレットハンドルのカラーリストから登録されているカラーのカラーコードを計算し、vListPtr で指定された配列に格納する。
vListPtr として 0 を指定すると、ヒープ上に配列を確保する。
再配置が発生する。

Cの関数► VList **CLPaletToValue*(Palet **paletHdl, VList *vListPtr);

返り値はカラーレコードの配列へのポインタ。
エラーが発生した場合、NULL が返る。

$A61F　CLSetScanEntry　3.0

引　数► long　paletHdl　; パレットハンドル
long　rgbScanProc　; カラースキャンルーチンへのポインタ

返り値► D0.L　前のカラースキャンルーチンへのポインタ
= -1　エラー
A0.L　前のカラースキャンルーチンへのポインタ

機　能► paletHdl で指定したパレットハンドルに rgbScanProc で指定したカラースキャンルーチンを登録する。
カラースキャンルーチンの仕様は以下のとおり。

int ScanRGB(cRGB *dRGB cRGB *sRGB, Palet **paletHdl)

引数

cRGB　*dRGB　実際の色を格納する RGB レコードへのポインタ
cRGB　*sRGB　求める色を格納する RGB レコードへのポインタ
Palet　**paletHdl　色を探す環境となるパレットハンドル

返り値

sRGB を意味するカラーコード

カラースキャンルーチンは、sRGB に最も近い色を paletHdl で指定したパレットハンドルのカラーリストのなかから探して dRGB に格納する。また、それをカラーコードに変換したものを返り値とする。イメージのコピーを行う際に呼び出される。

Cの関数► RGBScan *CLSetScanEntry*(Palet **paletHdl,
RGBScan (*rgbScanProc)());
返り値は前のカラースキャンルーチンへのポインタ。
エラーが発生した場合、NULL が返る。

ライブラリ　CLActive2

引　数► Palet　**paletHdl　; パレットハンドル

返り値► 0　パレットの割り付けに変更がなかった
0 以外　パレットの割り付けの変更があった

機　能► $A60C CLActive をコールし、自分以外のパレットに割り付けの変更が発生した場合は割り付け変更イベントを登録する。

Cの関数► int *CLActive2*(Palet **paletHdl);

ライブラリ　CLAddDevice

引　数► Palet　**paletHdl　　;パレットハンドル
int (*DevSet)() dSet　;カラーコードセットルーチン
int　cDevID　　;デバイスID

返り値► D0.L　登録したデバイスID
= -1　エラー

機　能► paletHdl で指定したパレットハンドルを、cDevID で指定したデバイス ID を持つデバイスパレットとしてシステムに登録する。
cDevID として-1 を指定すると、新しいデバイス ID が割り当てられる。
dSet には、デバイスパレットにカラーコードをセットするルーチンのエントリを指定する。割り付けに際してカラーコードを設定する必要がある場合に、このルーチンが呼ばれる。NULL を指定すると、割り付け時に何も呼び出さない。
DevSet 関数の仕様は以下のとおり。

```
typedef int (*DevSet)();
int DevSet(CInfo *ciPtr, int clNum);
```

CInfo	ciPtr	value および RGB 値が格納されている cInfo 構造体へのポインタ
int	clNum	デバイスのカラーリスト番号

SX コールとライブラリでは引数の数が違う。アセンブラから利用する場合は、デバイスセットルーチンのエントリを pHdl の dSet にセットすること。また、SXCALL 宣言によって実装されている callno ヘッダの$A613 CLAddDevice 関数は引き数が 2 つで、アセンブラからの場合と同様に呼び出す必要がある。

Cの関数► int *CLAddDevice*(Palet **paletHdl, DevSet (*dSet)(), int cDevID);

ライブラリ　CLDisposePalet2

引　数► Palet　**paletHdl　　;パレットハンドル

返り値► $A605 CLDisposePalet のリザルトコード。
CLDisposePalet が失敗した場合は、そのあとに実行した$A35A TSPostEventTsk のリザルトコード。

機　能► CLDisposePalet をコールし、自分以外のパレットに割り付けの変更が発生した場合は割り付け変更イベントを登録する。

Cの関数► int *CLDisposePalet2*(Palet **paletHdl);

ライブラリ　CLEvent

引　数► Window　*winPtr　　　；ウィンドウレコードへのポインタ
Palet　**paletHdl　　；パレットハンドル
TsEvent *eventRecPtr　；タスクマンイベントレコードへのポインタ

返り値► = 0　パレットの割り付けに変更がなかった
≠ 0　パレットの割り付けに変更があった

機　能► winPtr で指定したウィンドウに用いられている paletHdl で指定したパレットハンドルについて、eventRecPtr で指定されたイベントレコードの内容に対応する。
対応するのは、以下の 2 つのイベント。

- アクティベートイベント
- System1 イベント（タスクマンイベントコード 92 ： 割り付け変更イベント）

それぞれ、必要に応じて$A60C CLActive、$A60D CLRealloc を呼び出し、自分以外のパレットに割り付けの変更が発生した場合は割り付け変更イベントを登録する。
内部で$A35A TSPostEventTsk をコールしているため、タスクの切り替えが発生する。

Cの関数► int *CLEvent*(Window *winPtr, Palet **paletHdl, TsEvent *eventRec);

ビデオマン

$A700　VMInit　3.0

引　数► long　resNameListPtr　；リソースファイル名のリストへのポインタ

返り値► D0.L

0	正常終了
-1	エラー
other	すでに初期化済み・バージョン番号

A0.L

0	正常終了
-1	エラー
other	リソースマップへのハンドルのリストへのポインタ

機　能► ビデオマンを初期化する。resNameListPtr で指定した 6 個までのリソースファイルをビデオマンのローカルリソースとしてオープンする。
ビデオマンは IVM.X が起動した時点で初期化済みなので、一般のアプリケーションを呼び出す必要はない。

Cの関数► long *VMInit*(char *resNameListPtr[]);
返り値は、バージョン番号、またはリソースマップへのハンドルのリストへのポインタ、またはリザルトコード。

$A701　VMTini　3.0

引　数► なし

返り値► D0.L

0	正常終了
-1	エラー

機　能► ビデオマンの終了処理を行う。ローカルリソースは廃棄される。
一般のアプリケーションを呼び出す必要はない。

Cの関数► long *VMTini*(void);
返り値はリザルトコード。

$A710　VMExpand　3.0

引　数► long　bitsHdl　；展開先のビッツへのハンドル
long　buffHdl　；圧縮データへのハンドル
long　rscID　；VM モジュールのリソース ID
long　sRectPtr　；展開する元の画像の範囲を意味するレクタングルレコードへのポインタ

long　dRectPtr　；展開後の画像の範囲を意味するレクタングルレコードへのポインタ
long　user　；VMモジュールへ渡すデータ

返り値► D0.L　リザルトコード
A0.L　ビッツへのハンドル

機　能► buffHdl で指定した圧縮画像を rscID で指定した VM モジュールを用いて bitsHdl で指定したビッツに展開する。その際、元画像の sRectPtr で指定した領域を dRectPtr で指定したサイズで拡大/縮小する。また、必要がある場合は user で VM モジュールにデータを渡すことができる。
bitsHdl に 0 を指定した場合、ビデオマンがビッツを作成する。
rscID に -1 を指定した場合、画像のフォーマットを自動的に判断して適当なモジュールを使用する。
sRectPtr, dRectPtr は 0 を指定することで省略可能。
再配置が発生する。

Cの関数► Bits ***VMExpand*(Bits **bitsHdl, Handle buffHdl, long rscID, Rect *sRectPtr, Rect *dRectPtr, long user);
返り値はビッツへのハンドル。
エラーが発生した場合、NULL が返る。

$A711　VMCompress　3.0

引　数► long　bitsHdl　；元の画像のビッツへのハンドル
long　buffHdl　；圧縮データが格納されるブロックへのハンドル
long　rscID　；VMモジュールのリソース ID
long　sRectPtr　；圧縮する元の画像の範囲を意味するレクタングルレコードへのポインタ
long　dRectPtr　；圧縮後の画像の範囲を意味するレクタングルレコードへのポインタ
long　user　；VMモジュールへ渡すデータ

返り値► D0.L　リザルトコード
A0.L　圧縮データが格納されたブロックへのハンドル

機　能► bitsHdl で指定したビッツの画像を rscID で指定した VM モジュールを用いて圧縮し、buffHdl で指定したブロックに格納する。その際、元画像の sRectPtr で指定した領域を dRectPtr で指定したサイズで拡大/縮小する。
また、必要な場合には、user で VM モジュールにデータを渡すことができる。
buffHdl で指定したブロックが十分な大きさを持たない場合は、拡張される。
また、0 を指定した場合、ビデオマンがヒープ上に作成する。
rscID に -1 を指定した場合、$A716 VMSetCurrentID で指定されたモジュールを使用する。
sRectPtr, dRectPtr は 0 を指定することで省略可能。

再配置が発生する。

Cの関数► Handle *VMCompress*(Bits **bitsHdl, Handle buffHdl, long rscID, Rect *sRectPtr, Rect *dRectPtr, long user);
返り値は、圧縮データが格納されたブロックへのハンドル。
エラーが発生した場合、NULL が返る。

$A712 VMExpDirect 3.0

引　数► long bitsHdl ; 展開先のビッツへのハンドル
long buffHdl ; 圧縮データへのハンドル
long rscID ; VM モジュールのリソース ID
long user ; VM モジュールへ渡すデータ

返り値► D0.L リザルトコード
A0.L ビッツへのハンドル

機　能► buffHdl で指定した圧縮画像を rscID で指定した VM モジュールを用いて、bitsHdl で指定したビッツに高速展開する。必要な場合には、user で VM モジュールにデータを渡すことができる。
エラーチェック等を省いているため、高速に展開できる反面、以下の 3 点を守らない場合には動作が保証されない。

- bitsHdl としてロックされたビッツへのハンドルを指定しなければならない。
- rscID として圧縮画像のフォーマットに適したモジュールの ID を指定しなければならない。
- 元の画像と展開先のビッツのサイズは同じでなければならない。

再配置が発生する。

Cの関数► void *VMExpDirect*(Bits **bitsHdl, Handle buffHdl, long rscID, long user);
返り値はない。

$A713 VMGetInfo 3.0

引　数► long rscID ; VM モジュールのリソース ID
long buffHdl ; 圧縮データへのハンドル
long namePtr ; ファイルネームへのポインタ (ASCIIZ 型)
long tblPtr ; 結果が返るバッファ (256 バイト) へのポインタ
long user ; VM モジュールへ渡すデータ

返り値► D0.L VM モジュールのリソース ID/リザルトコード

-1	サポートできない圧縮フォーマット
-2	展開途中でエラーが発生した

A0.L　結果が返るバッファへのポインタ

(A0).L	rscID	VMモジュールのリソースID
4(A0)	idName	モジュール名(32バイト)
36(A0)	idData	モジュールに渡されたデータ(128バイト)
164(A0)	extFName	サポートする拡張子(4バイト)
168(A0).W	forPhoto	VMGetInfoでモジュールのリソースIDとして-1を指定した際に検索の対象となるなら1
170(A0).W	forMovie	$A739 VMRegistSample可能ならば1
172(A0).W	withHeader	識別ヘッダがついていて、$A710 VMExpandでリソースIDとして-1を指定したときに検索の対象となるなら1
174(A0).W	fixedBounds	サポートする圧縮画像のレクタングルが固定サイズならば1

(以上は、必ず格納されるデータ。以下は、モジュールによっては格納されない場合がある)

176(A0).W	bitsTy7e	最適なスクリーンタイプ
178(A0).W	useBits	色表現に必要なビット数。フルカラーの場合は-1
180(A0).W	withPalette	パレットを持つかどうかのフラグ
182(A0)	baseRect	想定されているベース画面のレクタングル
190(A0)	picRect	実際に画像展開されるレクタングル
198(A0).L	expTime	展開に必要な時間(1/100秒単位) 十分に早い場合は0、不明な場合は−1
202(A0)	reserved	システム予約(38バイト)
240(A0)	user	モジュールに固有な領域(16バイト)

機　能► buffHdlで指定した圧縮画像をrscIDで指定したVMモジュールで解析した結果を、tblPtrのバッファに返す。
rscIDとして -1 を指定すると、圧縮画像のフォーマットとnamePtrで指定したファイルネームをもとに適当なモジュールを呼び出す。rscIDに値を指定する場合、namePtrはNULLでもよい。
buffHdl, namePtrの両方にNULLを指定した場合、rscIDで指定したVMモジュールに関する情報(D0.L)だけが返される。
必要な場合には、userで VM モジュールにデータを渡すことができる。
再配置が発生する。

Cの関数► long *VMGetInfo*(long rscID,Handle buffHdl,char *namePtr, VideoIPtr tblPtr, long user);
返り値は、VMモジュールのリソースID、または結果を意味する数値。

$A714　VMRscInfo　3.0

引　数► long　rscType　; リソースタイプ
long　rscID　; リソース ID
long　listHdl　; 結果が返るブロックへのハンドル

返り値► D0.L　リザルトコード
A0.L　結果が格納されたブロックへのハンドル

機　能► このコールを呼び出したタスクがアクセス可能なリソースとビデオマンのリソース中から rscType で指定したリソースを探し、結果を listHdl で指定されたブロックに格納する。
rscID は意味を持たない。
listHdl で指定したブロックのサイズが足りない場合、ビデオマンが拡張する。また、listHdl として 0 を指定すると、ビデオマンがヒープ上に作成する。
rscType として−1 を指定すると、'VMIF', 'VMEC' の一覧を返す。
結果は以下のとおり。

```
list:
        ds.l  1    * データの個数
                   1 個目のデータ
        ds.l  1    * リソースタイプ
        ds.l  1    * リソース ID
                   2 個目のデータ
              ⋮
```

再配置が発生する。

Cの関数► long *VMRscInfo*(long rscType,long rscID,Handle listHdl);
返り値はリザルトコード。

$A715　VMRscHdlGet　3.0

引　数► long　rscType　; リソースタイプ
long　rscID　; リソース ID

返り値► D0.L　リザルトコード
A0.L　指定されたリソースへのハンドル

機　能► このコールを呼び出したタスクがアクセス可能なリソースとビデオマンのリソース中から rscType と rscID で指定したリソースを読み込み、そのハンドルを返す。
再配置が発生する。

Cの関数► Handle *VMRscHdlGet*(long rscType, long rscID);
返り値は指定されたリソースへのハンドル。

エラーが発生した場合、NULL が返る。

$A716 VMSetCurrentID 3.0

引　数► long rscID ; VM モジュールのリソース ID

返り値► D0.L 前の VM モジュールのリソース ID

機　能► 画像圧縮時にデフォルトで用いられる VM モジュールのリソース ID をセットする。

再配置が発生する。

Cの関数► long *VMSetCurrentID*(long rscID);

返り値は前の VM モジュールのリソース ID。

$A717 VMGetCurrentID 3.0

引　数► なし

返り値► D0.L デフォルトの VM モジュールのリソース ID

機　能► 画像圧縮時にデフォルトで用いられる VM モジュールのリソース ID を返す。

Cの関数► long *VMGetCurrentID*(void);

返り値は、デフォルトの VM モジュールのリソース ID。

$A718 VMGetPalette 3.0

引　数► long rscID ; VM モジュールのリソース ID

long buffHdl ; 圧縮データへのハンドル

long user ; VM モジュールへ渡すデータ

返り値► D0.L リザルトコード

A0.L パレットハンドル

機　能► buffHdl で指定したブロックに格納されている圧縮データを rscID で指定した VM モジュールで解析し、カラーパレットハンドルを返す。

rscID として -1 を指定すると、適当なモジュールを呼び出す。

必要な場合には、user で VM モジュールにデータを渡すことができる。再配置が発生する。

Cの関数► Palet ***VMGetPalette*(long rscID, Handle buffHdl, long user);

返り値はパレットハンドル。

エラーが発生した場合、NULL が返る。

$A730　VMSetAnim　3.0

引　数► long　animHdl　; アニメーションポートへのハンドル

返り値► D0.L　リザルトコード

A0.L　前のアニメーションポートへのハンドル

= 0　存在しない

機　能► animHdl で指定したアニメーションポートをカレントにする。

Cの関数► Handle *VMSetAnim*(Handle animHdl);

返り値は、前のアニメーションポートへのハンドル。

エラーが発生した場合、NULL が返る。

$A731　VMGetAnim　3.0

引　数► なし

返り値► D0.L　リザルトコード

= -1　存在しない

A0.L　アニメーションポートへのハンドル

= 0　存在しない

機　能► カレントアニメーションポートを返す。

Cの関数► Handle *VMGetAnim*(void);

返り値はアニメーションポートへのハンドル。

エラーが発生した場合、NULL が返る。

$A732　VMSetParam　3.0

引　数► long　parmID　; パラメータ ID

0(PARAM_DURATION)	総再生時間
1(PARAM_TIMESCALE)	タイムスケール
2(PARAM_RATE)	スピードレート
3(PARAM_CURTIME)	現在時刻
4(PARAM_SELTIME)	部分再生開始時刻
5(PARAM_SELDUR)	部分再生時間長
6(PARAM_PREFRATE)	デフォルトレート
7(PARAM_PLAYRECT)	再生レクタングル
8(PARAM_USERATOM)	ユーザデータ

long　value1　; 設定値 1

⋮

返り値► D0.L

0	正常終了
-1	エラー

A0.L　前の設定値

機　能► parmID で指定した内容について、各種値の設定を行う。

詳細は、各設定についてのマクロを参照のこと。

Cの関数► long *VMSetParam*(long parmID, long value);

返り値は前の設定値。

$A733　VMGetParam　3.0

引　数► long　parmID　; パラメータ ID

0(ARAM_DURATION)	総再生時間
1(ARAM_TIMESCALE)	タイムスケール
2(ARAM_RATE)	スピードレート
3(ARAM_CURTIME)	現在時刻
4(ARAM_SELTIME)	部分再生開始時刻
5(ARAM_SELDUR)	部分再生時間長
6(ARAM_PREFRATE)	デフォルトレート
7(ARAM_PLAYRECT)	再生レクタングル
8(ARAM_USERATOM)	ユーザデータ
$1000(PARAM_WIDTHHEIGHT)	アニメーション画像の縦横幅
$1001(PARAM_STATUS)	アニメーションポートの状態
$1002(PARAM_MEDIAMODE)	画像メディアの種類
$1003(PARAM_FRAMDUR)	指定フレームの再生時間
$1004(PARAM_TOTALSAMPLE)	総サンプル数
$1005(PARAM_CURSAMPLE)	カレントのサンプル ID
$1006(PARAM_TOTALFRAME)	総フレーム数

返り値► D0.L

0	正常終了
-1	エラー

A0.L　　前の設定値

機　能► parmIDで指定した内容について、設定されている値を返す。
詳細は、各設定についてのマクロを参照のこと。

Cの関数► long *VMGetParam*(long parmID);
返り値は前の設定値。

$A734　VMCreate　3.0

引　数► なし

返り値► D0.L　リザルトコード
A0.L　アニメーションポートへのハンドル

機　能► アニメーションポートをヒープ上にメモリモードで作成する。作成したアニメーションポートはオープン状態でカレントになる。
再配置が発生する。

Cの関数► Handle *VMCreate*(void);
返り値はアニメーションポートへのハンドル。
エラーが発生した場合、NULLが返る。

$A735　VMCreateF　3.0

引　数► long　fileID　;ファイルハンドル

返り値► D0.L　リザルトコード
A0.L　アニメーションポートへのハンドル

機　能► アニメーションポートをヒープ上にファイルモードで作成する。作成したアニメーションポートはオープン状態で、カレントになる。
内容を変更する場合は、ファイルをread/writeモードでオープンしておく必要がある。
すでに存在するファイルのファイルハンドルを渡した場合は、それがアニメーションファイルであるかどうかをチェックし、そうでない場合はエラーを返す。
アニメーションファイルの場合は変更モードとなり、モーション部をメモリに読み込む。
サイズが0のファイルのファイルハンドルを渡した場合は、ヘッダを書き込む。
再配置が発生する。

Cの関数► Handle *VMCreateF*(long fileID);
返り値はアニメーションポートへのハンドル。
エラーが発生した場合、NULLが返る。

$A736　VMOpen　3.0

引　数► long　animHdl　; アニメーションポートへのハンドル
返り値► D0.L　リザルトコード
機　能► animHdl で指定したクローズ状態のアニメーションポートをオープン状態にして、カレントにする。
再配置が発生する。
Cの関数► long *VMOpen*(Handle animHdl);
返り値はリザルトコード。

$A737　VMClose　3.0

引　数► long　animHdl　; アニメーションポートへのハンドル
返り値► D0.L　リザルトコード
機　能► animHdl で指定したオープン状態のアニメーションポートをクローズ状態にする。
メモリモードでオープンしている場合は、ヒープ上のデータは廃棄されない。
ファイルモードでオープンしている場合は、アニメーションファイルはクローズされない。モーション部、メディア部に変更があった場合は、それぞれをファイルに書き出す。
再配置が発生する。
Cの関数► long *VMClose*(Handle animHdl);
返り値はリザルトコード。

$A738　VMDispose　3.0

引　数► long　animHdl　; アニメーションポートへのハンドル
返り値► D0.L　リザルトコード
機　能► アニメーションポートを廃棄する。再生中の場合、再生は停止される。カレントアニメーションポートを廃棄した場合は、カレントアニメーションポートが存在しない状態になる。
メモリモードでオープンしている場合は、ヒープ上のデータは廃棄される。
ファイルモードでオープンしている場合は、ファイルはクローズされない（ファイルのクローズはアプリケーション側で行う必要がある）。モーション部に変更があった場合でもファイルへの書き出しは行わない。
再配置が発生する。
Cの関数► long *VMDispose*(Handle animHdl);
返り値はリザルトコード。

$A739　VMRegistSample　3.0

引　数► long　buffHdl　;(圧縮) 画像データへのハンドル
long　buffSize　;(圧縮) 画像データのバイト数
long　rscID　; VM モジュールのリソース ID
long　sizePt　; 画像の大きさを意味するポイント

返り値► D0.L　サンプル ID/リザルトコード

機　能► カレントアニメーションポート（オープン状態）に buffHdl で指定した画像データを 1 枚登録する。登録に成功した場合、登録画像の通し番号（サンプル ID）が返る。サンプル ID は$A738 VMDispose するまで有効で、のちに $A735 VMCreateF 等でオープンした場合は、画像とサンプル ID の対応関係は変化している場合がある。
rscID として-1 を指定すると、画像データのフォーマットから判断して適当な VM モジュールを呼び出す。
buffHdl は疑似ハンドルも可。
再配置が発生する。

Cの関数► long *VMRegistSample*(Handle buffHdl, long buffSize,
long rscID, LPoint sizePt);
返り値は、サンプル ID またはリザルトコード。

$A73A　VMDeleteSample　3.0

引　数► long　sampleID　; 削除する画像のサンプル ID

返り値► D0.L　リザルトコード

機　能► オープン状態のメモリモードのカレントアニメーションポートから sampleID で指定する画像を削除する。
再配置が発生する。

Cの関数► long *VMDeleteSample*(long sampleID);
返り値はリザルトコード。

$A73B　VMReferSample　3.0

引　数► long　sampleID　; サンプル ID
long　buffPtr　; 結果が返るバッファ (28 バイト) へのポインタ

返り値► D0.L　リザルトコード
A0.L　バッファへのポインタ

(A0).L	mediaMode	0 メモリモード 1 ファイルモード·編集中 2 ファイルモード·参照中
4(A0).L	fd	ファイルハンドル（メモリモードの場合は-1）
8(A0).L	ofstHdl	画像格納バッファへのハンドル
12(A0).L	size	画像データのバイト数
16(A0)	picRect	画像データのレクタングル
24(A0).L	rscID	画像データを扱うVMモジュールのリソースID

機　能► カレントアニメーションポートに登録されている sampleID で指定される画像に関する情報を buffPtr で指定されるバッファに返す。
これによって得られた画像データへのハンドルを利用して、画像の内容を変更してはならない。再配置が発生する。

Cの関数► long *VMReferSample*(long sampleID,VideoSampleIPtr buffPtr);
返り値はリザルトコード。

$A73C　VMGetSample　3.0

引　数► long　sampleID　；サンプル ID

返り値► D0.L　画像データのバイト数/リザルトコード
A0.L　画像データへのハンドル

機　能► カレントアニメーションポートに登録されている sampleID で指定される画像を、ヒープ上に再配置可能ブロックを作成して格納し、そのハンドルを返す。
再配置が発生する。

Cの関数► Handle *VMGetSample*(long sampleID);
返り値は画像データへのハンドル。
エラーが発生した場合、NULL が返る。

$A73D　VMGetBits　3.0

引　数► long　sampleID　；サンプル ID

返り値► D0.L　リザルトコード
A0.L　ビッツへのハンドル

機　能► カレントアニメーションポートに登録されている sampleID で指定される画像を、ビッツを作成して格納し、そのハンドルを返す。
再配置が発生する。

Cの関数► Bits ***VMGetBits*(long sampleID);
返り値はビッツへのハンドル。
エラーが発生した場合、NULL が返る。

$A73E　VMInsertFrame　3.0

引　数► long　sampleID　; サンプル ID
long　loc　; 挿入位置のフレーム番号
long　duration　; 表示時間
返り値► D0.L　リザルトコード
機　能► カレントアニメーションポートに登録されている sampleID で指定される画像を、loc で指定される位置のフレームの前に duration の表示時間を持たせて挿入する。
loc の範囲は 0 ≦ loc ≦ (枚数-1) で、-1 は末尾を意味する。
duration の単位は tv(1/TimeScale 秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する)。
総再生時間がマクロ VMSetDuration で指定した時間を超える場合でもエラーにはならないが、総再生時間を超過した部分は表示されない。
1 つの画像を何度指定してもかまわない。
最初に画像を登録する場合、loc として-1 を指定する。
再配置が発生する。
Cの関数► long *VMInsertFrame*(long sampleID,long loc,long duration);
返り値はリザルトコード。

$A73F　VMDeleteFrame　3.0

引　数► long　loc　; 挿入位置のフレーム番号
返り値► D0.L　削除されたフレーム番号/リザルトコード
機　能► カレントアニメーションポートに登録されている loc で指定されたフレームを削除する。
再配置が発生する。
Cの関数► long *VMDeleteFrame*(long loc);
返り値は、削除されたフレーム番号またはリザルトコード。

$A740　VMTimeToFrame　3.0

引　数► long　chkTime　; 該当するフレーム番号を調べる時刻
返り値► D0.L　フレーム番号/リザルトコード
A0.L　該当するフレームの表示開始時刻
機　能► chkTime で指定する時刻に表示中となるフレームの番号を返す。
chkTime の単位は tv(1/TimeScale 秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する)。
Cの関数► long *VMTimeToFrame*(long chkTime, long *frameStartTime);
long 型の変数 frameStartTime にフレームの表示開始時刻が格納される。

返り値は、フレーム番号またはリザルトコード。

$A741 VMFrameToSample 3.0

引　数► long frameNo ；フレーム番号

返り値► D0.L サンプルID/リザルトコード

機　能► カレントアニメーションポートでアニメーション再生する際に、frameNoで指定されるフレームで表示される画像のサンプルIDを返す。

Cの関数► long *VMFrameToSample*(long frameNo);
返り値は、サンプルIDまたはリザルトコード。

$A742 VMTrans 3.0

引　数► long sAnimHdl ；転送元アニメーションポートへのハンドル
long dAnimHdl ；転送先アニメーションポートへのハンドル

返り値► D0.L リザルトコード

機　能► sAnimHdlで指定されるアニメーションポートからdAnimHdlで指定されるアニメーションポートへ、内容をコピーする。主として、メモリモードのアニメーションポートの内容をディスクモードのアニメーションポートへコピー（セーブ）、またはその逆（ロード）に使用する。
転送先がファイルモードの場合、ファイルをread/writeモードでオープンしておく必要がある。
メモリ不足、ディスクへのアクセス失敗等の理由でコピーに失敗した場合のみ、転送先のアニメーションポートは破壊され、リザルトコードとしてER_TRANS($FFFFFB11 = -1263)が返る。
画像データとサンプルIDの対応関係は変化しない。メモリ⇒ファイルの転送の際、フレームに対応する画像が削除されていた場合には、転送先には0バイトの画像データが格納される。
再配置が発生する。

Cの関数► long *VMTrans*(Handle sAnimHdl, Handle dAnimHdl);
返り値はリザルトコード。

$A743 VMPlay 3.0

引　数► long animHdl ；アニメーションポートへのハンドル
long rectPtr ；レクタングルレコードへのポインタ（ローカル座標系）

返り値► D0.L リザルトコード

機　能► カレントビットマップのrectPtrで指定した領域に、animHdlで指定したア

ニメーションポートの内容の再生を開始する。
アニメーションポートはクローズ状態でなければならない。
画像データのレクタングルとrectPtrの大きさが異なる場合は拡大/縮小を行う。
再配置が発生する。

Cの関数► long *VMPlay*(Handle animHdl, Rect *rectPtr);
返り値はリザルトコード。

$A744 VMEvent 3.0

引　数► long animHdl ;アニメーションポートへのハンドル
long tsEventRec ;タスクマンイベントレコードへのポインタ

返り値► D0.L アニメーションの状態/リザルトコード

0	アニメーション中
1	終了

機　能► アイドルイベント等で呼び出すことによって、animHdlで指定したアニメーションポートの内容のアニメーション処理を行う。tsEventRecには、そのときのイベントレコードへのポインタを指定する。
アニメーションを再生するグラフレコードをカレントにしておく必要がある。
再配置が発生する。

Cの関数► long *VMEvent*(Handle animHdl, TsEvent *tsEventRec);
返り値は、アニメーションの状態またはリザルトコード。

$A745 VMStop 3.0

引　数► long animHdl ;アニメーションポートへのハンドル

返り値► D0.L リザルトコード

機　能► animHdlで指定したアニメーションポートのアニメーション再生を停止する。

Cの関数► long *VMStop*(Handle animHdl);
返り値はリザルトコード。

$A746 VMPause 3.0

引　数► long animHdl ;アニメーションポートへのハンドル
long pauseLvl ;ポーズレベルの加算値

返り値► D0.L 前のポーズレベル

機　能► 現在のポーズレベルにpauseLvlで指定した値を加え、結果が負である場合、animHdlで指定したアニメーションポートのアニメーション再生を一時停止する。ポーズレベルのデフォルト値は0。

pauseLvl として 0 を指定すると、現在のポーズレベルを返す。
Cの関数► long *VMPause*(Handle animHdl, long pauseLvl);
返り値は前のポーズレベル。

$A747 VMUpdate 3.0

引 数► long animHdl ; アニメーションポートへのハンドル
返り値► D0.L リザルトコード
機 能► animHdl で指定したアニメーションポートのアニメーション再生において、現在表示中の画像を再表示する。表示中の画像がない場合は-1 が返る。
再配置が発生する。
Cの関数► long *VMUpdate*(Handle animHdl);
返り値はリザルトコード。

$A748 VMSetUser 3.0

引 数► long dataHdl ; 設定するデータが格納されているブロックへのハンドル
返り値► D0.L リザルトコード
A0.L アニメーションポートに設定されたデータへのハンドル
機 能► カレントアニメーションポートのユーザデータとして、dataHdl で指定したブロックに格納されている情報を設定する。それ以前に設定されていたユーザデータのブロックは廃棄される。
dataHdl として NULL を指定した場合、ユーザデータを削除する。
再配置が発生する。
Cの関数► long *VMSetUser*(Handle dataHdl);
返り値はリザルトコード。

$A749 VMGetUser 3.0

引 数► なし
返り値► D0.L リザルトコード
A0.L アニメーションポートに設定されているユーザデータへのハンドル
機 能► カレントアニメーションポートのユーザデータとして設定されているブロックのハンドルを返す。
ブロックの内容の変更やブロックのサイズの変更は可能だが、廃棄してはならない。変更後は、そのハンドルを指定して$A748 VMSetUser しなければならない。
Cの関数► Handle *VMGetUser*(void);

返り値はユーザデータへのハンドル。

$A74A　VMDisplay　3.0

引　数► long　animHdl　;アニメーションポートへのハンドル
返り値► D0.L　リザルトコード
機　能► animHdl で指定したアニメーションポートがアニメーション再生中あるいはクローズ状態である場合、現在時刻に表示すべき画像を表示する。現在表示中の画像と同じである場合は表示しない。
再配置が発生する。
Cの関数► long *VMDisplay*(Handle animHdl);
返り値はリザルトコード。

マクロ　VMSetDuration

定　義►
```
#define VMSetDuration(dur)   VMSetParam(PARAM_DURATION, dur)
VMSetDuration   macro     dur
                move.l    dur,-(sp)
                move.l    #0,-(sp)    * PARAM_DURATION
                SXCALL    $A732       * VMSetParam
                addq.l    #8,sp
                endm
```
引　数► long　dur　;総再生時間
返り値► D0.L　負の値の場合はエラー
A0.L　前の設定値
機　能► グローバル系における総再生時間を dur にする。単位はtv(1/TimeScale秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する)。

マクロ　VMSetTimeScale

定　義►
```
#define VMSetTimeScale(scal)   VMSetParam(PARAM_TIMESCALE, scal)
VMSetTimeScale  macro     scal
                move.l    scal,-(sp)
                move.l    #1,-(sp)    * PARAM_TIMESCALE
                SXCALL    $A732       * VMSetParam
                addq.l    #8,sp
                endm
```
引　数► long　scal　;タイムスケール
返り値► D0.L　負の値の場合はエラー

AO.L　　前の設定値

機　能► グローバル系におけるタイムスケールを scal にする。これによって、時間の単位である tv（Time Value）が 1/TimeScale 秒となる。

マクロ　VMSetRate

定　義►
```
#define VMSetRate(rate)        VMSetParam(PARAM_RATE, rate)
VMSetRate       macro          rate
                move.l         rate,-(sp)
                move.l         #2,-(sp)      * PARAM_SETRATE
                SXCALL         $A732         * VMSetParam
                addq.l         #8,sp
                endm
```

引　数► long　　rate　　　　　；スピードレート

返り値► DO.L　　負の値の場合はエラー

AO.L　　前の設定値

機　能► グローバル系におけるスピードレートを rate にする。

マクロ　VMSetCurTime

定　義►
```
#define VMSetCurTime(tv)       VMSetParam(PARAM_CURTIME, tv)
VMSetCurTime       macro       tv
                   move.l      tv,-(sp)
                   move.l      #3,-(sp)    * PARAM_CURTIME
                   SXCALL      $A732       * VMSetParam
                   addq.l      #8,sp
                   endm
```

引　数► long　　tv　　　　　；現在時刻

返り値► DO.L　　負の値の場合はエラー

AO.L　　前の設定値

機　能► グローバル系における現在時刻を tv にする。単位は tv(1/TimeScale 秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する）。

マクロ　VMSetPlayRect

定　義►
```
#define VMSetPlayRect(rectPtr)VMSetParam(PARAM_PLAYRECT,rectPtr)
VMSetPlayRect      macro       rectPtr
                   move.l      rectPtr,-(sp)
                   move.l      #7,-(sp)        * PARAM_PLAYRECT
```

```
                SXCALL    $A732       * VMSetParam
                addq.l    #8,sp
                endm
```

引　数► long　rectPtr　；レクタングルレコードへのポインタ

返り値► D0.L　負の値の場合はエラー

A0.L　前のレクタングルレコードへのポインタ

機　能► 画像の再生レクタングルを rectPtr で指定したレクタングルにする。

マクロ　VMSetUserAtom

定　義► #define VMSetUserAtom(hdl)　VMSetParam(PARAM_USERATOM,hdl)

```
VMSetUserAtom   macro     hdl
                move.l    hdl,-(sp)
                move.l    #8,-(sp)    * PARAM_USERATOM
                SXCALL    $A732       * VMSetParam
                addq.l    #8,sp
                endm
```

引　数► long　hdl　；ユーザデータへのハンドル

返り値► D0.L　負の値の場合はエラー

A0.L　前のユーザデータへのハンドル

機　能► ユーザデータへのハンドルを設定する。

マクロ　VMGetDuration

定　義► #define VMGetDuration()　VMGetParam(PARAM_DURATION)

```
VMGetDuration   macro
                move.l    #0,-(sp)   * PARAM_DURATION
                SXCALL    $A733      * VMGetParam
                addq.l    #4,sp
                endm
```

引　数► なし

返り値► D0.L　負の値の場合はエラー

A0.L　総再生時間

機　能► グローバル系における総再生時間を返す。単位は tv(1/TimeScale 秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する)。

マクロ　VMGetTimeScale

定　義► #define VMGetTimeScale()　VMGetParam(PARAM_TIMESCALE)

```
VMGetTimeScale    macro
                  move.l    #1,-(sp)   * PARAM_TIMESCALE
                  SXCALL    $A733      * VMGetParam
                  addq.l    #4,sp
                  endm
```

引　数► なし

返り値► D0.L　負の値の場合はエラー
A0.L　タイムスケール

機　能► グローバル系におけるタイムスケールを返す。

マクロ　VMGetRate

定　義►

```
#define VMGetRate()        (long)VMGetParam(PARAM_RATE)
VMGetRate       macro
                move.l      #2,-(sp)   * PARAM_RATE
                SXCALL      $A733      * VMGetParam
                addq.l      #4,sp
                endm
```

引　数► なし

返り値► D0.L　負の値の場合はエラー
A0.L　スピードレート

機　能► グローバル系におけるスピードレートを返す。

マクロ　VMGetCurTime

定　義►

```
#define VMGetCurTime()     VMGetParam(PARAM_CURTIME)
VMGetCurTime    macro
                move.l      #3,-(sp)   * PARAM_CURTIME
                SXCALL      $A733      * VMGetParam
                addq.l      #4,sp
                endm
```

引　数► なし

返り値► D0.L　負の値の場合はエラー
A0.L　現在時刻

機　能► グローバル系における現在時刻を返す。単位は tv(1/TimeScale 秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する)。

マクロ　VMGetPlayRect

定　義► #define VMGetPlayRect(rectPtr)VMGetParam(PARAM_PLAYRECT,rectPtr)

```
VMGetCurTime     macro    rectPtr
                 move.l   rectPtr,-(sp)
                 move.l   #7,-(sp)          * PARAM_PLAYRECT
                 SXCALL   $A733             * VMGetParam
                 addq.l   #8,sp
                 endm
```

引　数► long　rectPtr　；結果が返るレクタングルレコードへのポインタ

返り値► D0.L　負の値の場合はエラー
A0.L　再生レクタングルのレクタングルレコードへのポインタ

機　能► 画像の再生レクタングルを返す。

マクロ　VMGetUserAtom

定　義► #define VMGetUserAtom()　VMGetParam(PARAM_USERATOM)

```
VMGetUserAtom    macro
                 move.l   #8,-(sp)   * PARAM_USERATOM
                 SXCALL   $A733      * VMGetParam
                 addq.l   #4,sp
                 endm
```

引　数► なし

返り値► D0.L　負の数の場合はエラー
A0.L　ユーザデータへのハンドル

機　能► ユーザデータへのハンドルを返す。

マクロ　VMGetWidthHeight

定　義► #define VMGetWidthHeight()(point_t)VMGetParam(PARAM_WIDTHHEIGHT)

```
VMGetWidthHeight macro
                 move.l   #$1000,-(sp)   * PARAM_WIDTHHEIGHT
                 SXCALL   $A733          * VMGetParam
                 addq.l   #4,sp
                 endm
```

引　数► なし

返り値► D0.L　負の数の場合はエラー
A0.L　アニメーション画像の縦横幅を意味するポイント

機　能► アニメーション画像の縦横幅を返す。

マクロ VMGetStatus

定　義►
```
#define VMGetStatus()   VMGetParam(PARAM_STATUS)
VMGetStatus     macro
                move.l      #$1001,-(sp)    * PARAM_STATUS
                SXCALL      $A733           * VMGetParam
                addq.l      #4,sp
                endm
```

引　数► なし

返り値► D0.L　負の数の場合はエラー

A0.L　アニメーションポートの状態

0	クローズ状態
1	オープン状態
2	アニメーション中

機　能► アニメーションポートの状態を返す。

マクロ VMGetMediaMode

定　義►
```
#define VMGetMediaMode()    VMGetParam(PARAM_MEDIAMODE)
VMGetMediaMode  macro
                move.l      #$1002,-(sp)    * PARAM_MEDIAMODE
                SXCALL      $A733           * VMGetParam
                addq.l      #4,sp
                endm
```

引　数► なし

返り値► D0.L　負の値の場合はエラー

A0.L　メディアの種類

0	メモリモード
1	ファイルモード（変更中）
2	ファイルモード

機　能► 画像メディアの種類を返す。

マクロ VMGetFrameDuration

定　義►
```
#define VMGetFrameDuration(frm) VMGetParam(PARAM_FRAMEDUR,frm)
VMGetFrameDuration      macro       frm
                        move.l      frm,-(sp)
                        move.l      #$1003,-(sp)    * PARAM_FRAMDUR
                        SXCALL      $A733           * VMGetParam
                        addq.l      #8,sp
```

```
                    endm
```

引　数► long　　frm　　　　　; フレーム番号

返り値► D0.L　　負の値の場合はエラー

　　　　A0.L　　再生時間

機　能► frm で指定したフレームの再生時間を返す。単位は tv(1/TimeScale 秒。$A732 VMSetParam/マクロ VMSetTimeScale で設定する)。

マクロ　VMGetTotalSample

定　義►
```
#define VMGetTotalSample()    VMGetParam(PARAM_TOTALSAMPLE)
VMGetTotalSample  macro
                  move.l   #$1004,-(sp)   * PARAM_TOTALSAMPLE
                  SXCALL   $A733          * VMGetParam
                  addq.l   #4,sp
                  endm
```

引　数► なし

返り値► D0.L　　負の値の場合はエラー

　　　　A0.L　　総サンプル ID 数

機　能► 総サンプル ID 数を返す。

マクロ　VMGetCurSample

定　義►
```
#define VMGetCurSample()    VMGetParam(PARAM_CURSAMPLE)
VMGetCurSample  macro
                move.l   #$1005,-(sp)   * PARAM_CURSAMPLE
                SXCALL   $A733          * VMGetParam
                addq.l   #4,sp
                endm
```

引　数► なし

返り値► D0.L　　負の値の場合はエラー

　　　　A0.L　　カレントのサンプル ID

機　能► カレントのサンプル ID を返す。

マクロ　VMGetTotalFrame

定　義►
```
#define VMGetTotalFrame()    VMGetParam(PARAM_TOTALFRAME)
VMGetTotalFrame  macro
                 move.l   #$1006,-(sp)   * PARAM_TOTALFRAME
                 SXCALL   $A733          * VMGetParam
```

```
        addq.l  #4,sp
        endm
```

引　数► なし

返り値► D0.L　　負の値の場合はエラー
A0.L　　総フレーム数

機　能► 総フレーム数を返す。

ライブラリ　VMSetParam

引　数► long　　paramID　　　　; 設定するパラメータの ID
⋮　　　　　　　; 設定値

返り値► 前の設定値

機　能► paramID で指定した内容について、値の設定を行う。
関数のプロトタイプ宣言では、第2引数で各種の型を処理できるように可変長引数を用いて宣言されているが、ライブラリでは引数を必ず2つ持つ関数として実装しているため、**実際に使用する場合はマクロを使用しなければならない。**

Cの関数► long *VMSetParam*(long paramID, ...);

ライブラリ　VMGetParam

引　数► long　　paramID　　　　; 参照するパラメータの ID

返り値► 前の設定値

機　能► paramID で指定した内容について、設定されている値を返す。
SX コールと引数が異なる場合がある。通常、引数は 1 つであるが、$A733 VMGetParam では、第1引数が PARAM_PLAYRECT ならびに PARAM_FRAMEDUR の場合のみ、2 つの引数が必要である。これらのパラメータについて値を参照する場合は、マクロ VMGetParam2 を用いなければならない。
実際に使用する場合は、マクロを使用しなければならない。

Cの関数► long *VMGetParam*(long paramID);

ライブラリ　VMGetParam2

引　数► long　　paramID　　　　; 参照するパラメータの ID
⋮

返り値► 前の設定値

機　能► paramID で指定した内容について設定されている値を返す。
この関数は、PARAM_PLAYRECT ならびに PARAM_FRAMEDUR の 2 つのパラメータの内容を参照する場合にかぎり、使用することが可能。
実際に使用する場合は、マクロを使用しなければならない。

Cの関数 ► `long` *VMGetParam2*`(long paramID, ...);`

SXコール番号順索引

INDEX SXコール名前順索引

あとがき

早いものでX68000が誕生してからもう9年近く。その半分はSX-WINDOWの歴史でもあります。この5年間、たくさんの人が、その人なりのやりかたでSX-WINDOWとつきあってきました。FSX.Xを解析する人、実用アプリケーションを作る人、小物ソフトでデスクトップを飾る人、環境のカスタマイズに命を賭ける人、開発環境を作る人。そしてそんな人たちとともにSX-WINDOWを支えてきたたくさんの人たち。この人たちの形作ってきた仮想的な「場」には、確かにひとつの文化が存在します。よくいえば「ユーザ主導」、ありていにいうならば「メーカのイニシアチブ不足」ゆえの文化ですが、これはこれで結構楽しい。……などといっているわたしたちのような人間がいるから、いつまでもメジャーになれないのかもしれませんが。

しかし、ひとつだけ確かなことは、ユーザの努力、そして、地道にサポートを続けてきたメーカの努力がこれだけの文化を花開かせたということは、日本のパソコン史のなかのひとつの奇跡であるということです。本書で提供するSX-WINDOWの開発環境や、CD-ROMに収められたフリーソフトはその奇跡のほんの一部でしかありません。なぜなら、奇跡はあなたが作ったSX-WINDOWアプリケーションが加わることで完全なものとなるからです。

本書という小さな奇跡にご協力いただいた皆さんに最上級の感謝を込めて。

フリーソフトウェアの収集ならびにその紹介に尽力していただいた葛生高司氏、LIBSXCのほとんどの改造を手掛けたNiggle氏、callno headerの作成を手伝ってくれた松森ひろき氏、数々の助言をいただき、また深夜の突然の訪問にも快く応じてくださった沖勝氏、多機能SCSIデバイスドライバSUSIEの添付を快諾してくださった後藤浩昭氏に感謝します。そして、uebee氏、lucas氏、ouzak氏の見返りを期待しない善意に感謝します。そのほか、書ききれないほどたくさんの人たちに、心から感謝いたします。

“己が窓を開かれよ。されば世界は己が手に。”

著者一同

著者略歴

吉沢正敏 (Yoz.)

『SX-WINDOW プログラミング』『追補版 SX-WINDOW プログラミング』の著者。いわずと知れた元祖 SXer。

牛島健雄 (Ussy)

自称マルチ (ぐーたら) クリエータ。プログラマからライターまでこなす謎の人物。

西田文彦 (KUM)

本書添付の LIBSXC など、これまでに数多くのシステム系 SX アプリケーションを発表している。これまた謎の多い人物。

小浜 純 (George)

SX-WINDOW 上の標準通信ソフトとまでいわれる QuTerm の作者として有名。そのヘビースモーカぶりは SXer 随一。

参考文献

『X68k Programming Series #1 X68000 Develop.』
吉野智興・中村祐一・石丸敏弘・今野幸義著　ソフトバンク刊　1993
『X68k Programming Series #2 X680x0 libc.』
村上敬一郎・大西恵司・萩野祐二著　ソフトバンク刊　1993
『X68k Programming Series #0 X680x0 Develop. & libc II』
吉野智興・中村祐一・石丸敏弘・今野幸義・村上敬一郎・大西恵司著　ソフトバンク刊　1994
『SX-WINDOW プログラミング』
吉沢正敏著　ソフトバンク刊　1991
『追補版 SX-WINDOW プログラミング』
吉沢正敏著　ソフトバンク刊　1991
『詳説 C 言語』
サミュエル・ハービソン／ガイ・スティール著　斉藤信男監訳　ソフトバンク刊　1989
『68000 プログラミング入門』
Tim King, Brian Knight 共著　鈴木 隆監訳　アスキー出版局刊　1984

「C MAGAZINE」 '93 年 11 月 & 12 月 “X68k 活用講座 SX-WINDOW プログラミング”

「SX-WINDOW 開発キット Workroom SX-68K SX ライブラリマニュアル」VOL.1,2　シャープ
「SX-WINDOW 開発キット Workroom SX-68k プログラマーズマニュアル」 シャープ
「SX-WINDOW 開発キット用ツール集 ユーザーズマニュアル」 シャープ
「X68000 C Compiler PRO-68k プログラマーズマニュアル」 シャープ

INDEX 用語索引

記号・数字

A

B

C

D

E

F

G

H

I

L

M

N

T

U

V

シ

ス

セ

ソ

タ

ツ

テ

ト

ナ

ヌ

ハ

ヒ

フ

ヘ

マ

メ

モ

ヨ

ラ

リ

レ

ロ

ワ

SX-WINDOW ver.3.1開発キット

1995年8月8日　初版第1刷発行

著　者　吉沢正敏・牛島健雄・西田文彦・小浜 純

発行者　橋本五郎

発行所　ソフトバンク株式会社　出版事業部

〒103　東京都中央区日本橋浜町3-42-3

販売　03 (5642) 8101

編集　03 (5642) 8140

装　丁　勝俣正希

表紙画　江口修平

印刷所　東京書籍印刷株式会社

Printed in Japan 1995.

ISBN4-89052-748-6 C0055

落丁、乱丁本は小社販売局にてお取り替え致します。

定価は表紙に表示してあります。

SX-WINDOW ver.3.1 開発キット

弊社ソフトバンクの本をお買い上げいただきありがとうございます。今後の編集の資料にさせていただきますので、下記のアンケートにお答え下さい。ご協力をお願いいたします。

■この本を何でお知りになりましたか？

1. 雑誌広告（雑誌名／　　　　　　　　　）
2. 雑誌の紹介記事で（雑誌名／　　　　　　　　　）
3. 書店で見て　　4. 書店ですすめられて　　5. 人にすすめられて
6. その他（　　　　　　　　　）

■この本をお買上げの書店名は？

都・道　府・県　　　　区・市　　　　書店

■以下の質問にお答え下さい

内　容	1. わかりやすい	2. ふつう	3. わかりにくい
装　丁	1. よい	2. ふつう	3. わるい
価　格	1. 高い	2. ふつう	3. 安い
CD-ROMドライブ		1. 持っている	2. 持っていない
通　信	1. している	2. していない	

■お読みになった感想をお聞かせ下さい

■今後どのような企画をお望みですか？

ISBN4-89052-748-6

C0055 P5800E

9784890527489

定価5,800円
(本体5,631円)

1910055058002

SX-WINDOW ver.3.1 開発キット

本書は、シャープ提供の開発環境「Workroom SX-68K」と、
『追補版SX-WINDOWプログラミング』などで提供されたフリーソフトによる
開発環境を統合し、SX-WINDOW ver.3.1の機能を利用した
アプリケーション開発環境を提供するものです。
添付FDには本書の著者たちが推奨する開発環境とCD-ROMドライバが、
添付CD-ROMには200本弱のSX-WINDOW対応フリーソフトを収録しています。
また、ver.3.1までのすべてのSXコールリファレンスをまとめてあります。